NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE
SYSTEM ANALYSIS AND PROTOCOL VERIFICATION
BY AN INTERACTIVE PETRI NET PACKAGE

by

XIAOLING QIU

A thesis
presented to the University of Ottawa
in partial fulfillment of the
requirements for the degree of
MASTER OF COMPUTER SCIENCE
in
Department of Computer Science

© Xiaoling Qiu, Ottawa, Canada, 1984.
The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.
ABSTRACT

In this thesis, I first review the fundamental theory and techniques of basic and extended Petri nets and describe their application in system analysis and verification of communication network protocols. My contribution in this part includes extending and refining the techniques of generating reachability trees from basic Petri nets to token-coloured and predicate/action Petri nets. I also improve techniques for the specification of some protocol components, such as timeout mechanisms and FIFO queues, using Petri net models.

The major work of this thesis is the development of PNPUO, a Petri Net Package developed at the University of Ottawa, as a tool for system analysis and protocol verification. PNPUO is an interactive system with the capability of storing, displaying, modifying, and analysing token-coloured and/or predicate/action Petri nets. It is implemented in PASCAL on an AMDAHL computer under the CMS Operating System. The package automates the analysis process of determining enumerative and structural properties of Petri nets, such as safeness, boundedness, conservation, deadlock freedom, livelock freedom, loops, place invariants, transition invariants, etc.
PNPUO can also be used to verify communication network protocols specified by Petri nets. As illustrations, we have applied it on two well-known protocol systems: the alternating bit protocol and the Connection and Disconnection Phases in Class 1 and Class 2 of the ECMA Transport Protocol. For the first protocol, a predicate/action Petri net is used; and for the second, a basic Petri net and a predicate/action Petri net are used. The analysis results indicate that the design of these two protocols are logically correct. We conclude that PNPUO is a very useful package.
ACKNOWLEDGEMENTS

The author wishes to express her deep sense of gratitude to her supervisor Dr. T. Y. Cheung for his valuable advice and outstanding supervision throughout this research, and his patient guidance during the writing of this dissertation. His extremely helpful instructions and encouragement are always appreciated.

Sincere thanks are also due to all her professors and colleagues, especially Dr. R. L. Probert, Dr. L. Logrippo, Dr. H. Ural and Dr. G. H. Masapati, for their kind supports and suggestions.

The author appreciates the important support from Beijing General Research Institute of Mining and Metallurgy (China), which granted her the leave of absence to carry out this stimulating research. She also owes a debt of thanks to her parents Mr. and Mrs. Y. Z. Qiu, her husband Mr. Z. D. Sun, and her children, Fang and Xiang, for their patience, perseverance and understanding.

The financial support from University of Ottawa and Natural Sciences and Engineering Research Council of Canada is acknowledged with gratitude.
# CONTENTS

**ABSTRACT** .................................................. iv

**ACKNOWLEDGEMENTS** .......................................... vi

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. SYSTEM ANALYSIS BY PETRI NETS</td>
<td>5</td>
</tr>
<tr>
<td>Basic Petri Nets</td>
<td>5</td>
</tr>
<tr>
<td>Definitions</td>
<td>6</td>
</tr>
<tr>
<td>Execution of PN</td>
<td>10</td>
</tr>
<tr>
<td>Extensions of Petri Nets</td>
<td>12</td>
</tr>
<tr>
<td>Token-coloured PN</td>
<td>13</td>
</tr>
<tr>
<td>Predicate/Action PN</td>
<td>15</td>
</tr>
<tr>
<td>Timed Petri Nets</td>
<td>17</td>
</tr>
<tr>
<td>Petri Nets with Inhibitor Arcs</td>
<td>18</td>
</tr>
<tr>
<td>Analysis by Petri Nets</td>
<td>19</td>
</tr>
<tr>
<td>Problems for Analysis</td>
<td>19</td>
</tr>
<tr>
<td>Techniques for Analysis</td>
<td>22</td>
</tr>
<tr>
<td>III. PROTOCOL SPECIFICATION AND VERIFICATION BY PETRI NETS</td>
<td>25</td>
</tr>
<tr>
<td>Specification and Verification of Protocols</td>
<td>26</td>
</tr>
<tr>
<td>Protocol Specification Using PN Models</td>
<td>29</td>
</tr>
<tr>
<td>Specification of Protocol Entities by PN</td>
<td>29</td>
</tr>
<tr>
<td>Specification of Virtual Media by PN</td>
<td>32</td>
</tr>
<tr>
<td>Specification of Recovery Mechanisms</td>
<td>38</td>
</tr>
<tr>
<td>Service Specification by PN Models</td>
<td>41</td>
</tr>
<tr>
<td>Protocol Verification Using Petri Nets</td>
<td>42</td>
</tr>
<tr>
<td>Logical Correctness</td>
<td>43</td>
</tr>
<tr>
<td>Conformance of Protocol with Service</td>
<td>44</td>
</tr>
<tr>
<td>Specification</td>
<td></td>
</tr>
<tr>
<td>IV. PNPuo - AN INTERACTIVE PETRI NET PACKAGE AT UNIVERSITY OF OTTWA</td>
<td>46</td>
</tr>
<tr>
<td>Objectives and Capabilities of PNPuo</td>
<td>46</td>
</tr>
<tr>
<td>User's Interface with System</td>
<td>48</td>
</tr>
<tr>
<td>Scope and Limitation of Application</td>
<td>50</td>
</tr>
<tr>
<td>Creating Modules for Analysis</td>
<td>51</td>
</tr>
<tr>
<td>Displaying of Data</td>
<td>56</td>
</tr>
<tr>
<td>Modifying Modules</td>
<td>57</td>
</tr>
</tbody>
</table>
V. PROTOCOL VERIFICATION BY PNPuo

Analysis of the Alternating Bit Protocol by PNPuo ........ 73
A Predicate/Action PN Model ...................... 74
Results of Analysis ................................ 76
Analysing the Connection and Disconnection Phases of Class 1 and Class 2 of Transport Protocol ........ 82
The BPN Model for Transport Protocol ................. 84
Results of Analysis Based on a BPN Model ............. 86
Results of Analysis Based on A Predicate/action PN .... 89

VI. CONCLUSION AND SUGGESTIONS FOR FURTHER RESEARCH AND DEVELOPMENT ........................................ 93

REFERENCES ............................................. 96

Appendix .................................................

A. PROGRAM LISTINGS OF PNPuo ................. 101
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>A Simple Petri Net</td>
<td>6</td>
</tr>
<tr>
<td>2-2</td>
<td>The Reachability Tree of the PN in Figure 2-1</td>
<td>12</td>
</tr>
<tr>
<td>2-3</td>
<td>A Token-Coloured PN for the Dining Philosophers Problem</td>
<td>15</td>
</tr>
<tr>
<td>2-4</td>
<td>An Example of Predicate/Action PN</td>
<td>17</td>
</tr>
<tr>
<td>2-5</td>
<td>A PN with An Inhibitor Arc</td>
<td>18</td>
</tr>
<tr>
<td>3-1</td>
<td>Service at Layer N</td>
<td>27</td>
</tr>
<tr>
<td>3-2</td>
<td>A TPN Model for the Alternating Bit Protocol</td>
<td>32</td>
</tr>
<tr>
<td>3-3</td>
<td>(a) A Perfect Channel (b) A Channel Allowing Message Loss</td>
<td>34</td>
</tr>
<tr>
<td>3-4</td>
<td>A Flow Control Model (a) For One MSG Type (b) For Two MSG Types</td>
<td>35</td>
</tr>
<tr>
<td>3-5</td>
<td>A'FIFO Channel Model (a) For One MSG Type (b) For Two MSG Types</td>
<td>36</td>
</tr>
<tr>
<td>3-6</td>
<td>Time-out Mechanism· (a) General Model (b) BPN Model</td>
<td>39</td>
</tr>
<tr>
<td>4-1</td>
<td>Control Hierarchy of User Interface</td>
<td>49</td>
</tr>
<tr>
<td>4-2</td>
<td>Program Organization of PNPUO</td>
<td>60</td>
</tr>
<tr>
<td>4-3</td>
<td>Flowchart of Procedure REACHABILITY</td>
<td>67</td>
</tr>
<tr>
<td>4-4</td>
<td>Petri Net for the Example</td>
<td>69</td>
</tr>
<tr>
<td>4-5</td>
<td>Reachability Tree of the PN in Figure 4-4</td>
<td>72</td>
</tr>
<tr>
<td>5-1</td>
<td>A Predicate/Action PN Model for the Alternating Bit Protocol</td>
<td>75</td>
</tr>
<tr>
<td>5-2</td>
<td>Reachability Tree for A Perfect Channel</td>
<td>77</td>
</tr>
</tbody>
</table>
5-3. The Reachability Tree When a MSG is Lost in the 1st state of Execution ................. 77
5-4. The Reachability Tree When a MSG is Lost in the 2nd state of Execution .................. 78
5-5. The Reachability Tree When a MSG is Lost in the 3rd State of Execution ................ 79
5-6. A BPN Model for the Connection and Disconnection Phases of Transport Protocol ........ 85
5-7. Reachability Tree of the PN in Figure 5-6 ........................................ 86
5-8. A Predicate/Action PN Model for the Transport Protocol ...................................... 90
5-9. The Reachability Tree for the improved Model with A=B=C=0 ............................. 91
5-10. The Reachability Tree for the Improved Model with A=1 and B=C=0 ...................... 91
5-11. The Reachability Tree for the Improved Model with A=B=1 and C=0 ..................... 92

LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Procedures and Data Types of Module CREATE</td>
<td>62</td>
</tr>
<tr>
<td>2. Procedures of Module DISPLAY</td>
<td>63</td>
</tr>
</tbody>
</table>
Chapter I

INTRODUCTION

Many real-life systems are very complicated to describe, design and implement. Since it is also extremely hard and expensive to correct errors once a system has been implemented, it is important to develop tools for describing and analysing these systems and detecting their errors at an early stage. Automation of these tools will provide invaluable design aids.

There exist many approaches for system modeling and analysis, such as finite state machines, Petri nets, abstract data types, high level languages, temporal logic, etc. [DAIZ82] [DANT80]. As one of the common abstract formal models, Petri nets (PN) are efficient for studying both static and dynamic properties of concurrent systems [MURA77] [RAMA82]. With their simple diagrams for description and powerful techniques for analysis, PN are also used in the modeling of software and hardware systems [NELS83] [PETE77].

In particular, communication network protocols are systems with a high degree of concurrency. Their specification, verification and testing have become important research topics recently. Petri nets have been
applied in these fields lately. Most of these works use basic Petri nets to model simple protocols [BERT82] [TANE81]. A few articles are devoted to the specification of complicated protocols using numerical Petri nets [BILL82] or to their verification using a special type of predicate/action Petri nets [COUR84] [JURG84].

As the size of the Petri nets increases, complexity of the model grows rapidly and analysis by manual processes becomes extremely long and tedious. It is useful to have a software tool for automating the processes. As far as we know, there exists only one such tool, called OGIVE [COUR84]. This system is commercially available only to a limited group of users in the business world. There are very limited publications, especially about its implementational and operational details. It can not handle Petri nets with predicates and actions. In this thesis, we develop another PN package for research purpose. It can handle Petri nets with more general types of predicates and actions, as well as the token-coloured Petri nets, which OGIVE cannot handle.

This thesis begins with a review of the underlying theory and techniques of several types of Petri nets. We then improve the technique of generating reachability trees. Our major work is the design and development of an interactive Petri net package, called PNPUO -- a Petri Net Package developed at the University of Ottawa, to be used as a
computer-aided design tool. This tool provides the users with the facilities of storing, displaying, modifying, and analyzing Petri nets. Using PNPUC, we have specified and analyzed two practical protocols, the alternating bit protocol and the Connection and Disconnection Phases in Classes 1 and 2 of the ECMA Transport Protocol.

Chapter 2 reviews the fundamental techniques and properties of basic Petri nets and many types of more powerful Petri nets extended from the basic one, namely token-coloured Petri nets, predicate/action Petri nets, timed Petri nets, Petri nets with inhibitor arcs, etc. We then extend the transition firing rules and the methodology for generating reachability trees from basic PN to token-coloured and predicate/action PN. This chapter also describes two techniques for analysis, namely reachability analysis and incidence analysis, by which one can derive the enumerative and structural properties of a Petri net.

A vast amount of research work has been done in the application of Petri nets to protocol modeling and verification. Chapter 3 reviews briefly some of these contributions related to our thesis. It also includes our study in the modeling of channels with flow control, FIFO queues for more than one message and the modeling of timeout mechanism with basic Petri nets.
Chapter 4 gives the implementation details, the system environment and user interface of PNPUO. It consists of four main modules: CREATE, DISPLAY, MODIFY and ANALYSIS. System analysis uses our extended method for generating reachability trees. We also present the application of PNPUO to a simple PN for the purpose of explaining the operational steps of this package. At the end of this chapter, we propose some aspects of improvements on PNPUO for further study.

Chapter 5 concentrates on the application of PNPUO to the verification of logical correctness of protocol systems. Two examples are considered: a data link protocol and a transport protocol. We develop a predicate/action PN model for the first protocol so that the 'time-out' function can be realized easily. For the second protocol, we use the same PN model as described in [BERT82]. Our analysis results confirm those obtained manually in [BERT82]. Our experience shows that PNPUO is an efficient package and useful for protocol verification. A program listing of PNPUO is attached in the Appendix.
Chapter II
SYSTEM ANALYSIS BY PETRI NETS

Since its inception in 1962 [PETR62] as an abstract formal model, Petri nets (PN) have been applied to many areas of system science. In this chapter, we explain the techniques of analysing a general system specified by means of Petri nets. We first present the basic terminology, properties and fundamental theory of Petri nets in Section 2.1. Different extensions from basic Petri nets are then given in Section 2.2. In particular, based on the theory of basic Petri nets, we develop the rules for firing transitions and the method for generating reachability trees for coloured PN and predicate/action PN. The techniques for system analysis by PN are explained in Section 2.3.

2.1 BASIC PETRI NETS

We first describe the components of and the operations on basic Petri nets. As shown in Figure 2-1, a Petri net is denoted as a directed graph with two types of nodes, circles (called places ) and bars (called transitions ). These nodes are connected by directed arcs from places to transitions and from transitions to places. A transition represents an event, while its input and output places
indicate the enabling and the resulting conditions of the occurrence of that event, respectively. Tokens, denoted by the existence of dots in the places, imply the fulfillment of the corresponding conditions.

![Petri Net Diagram]

Figure 2-1: A Simple Petri Net

Some formal definitions, concepts and properties about basic Petri nets [BERT82] [HOPC79] [MAGO84] [PETE77] [PETE81] [SYMO81] are given below.

2.1.1 Definitions

Definition 2-1

A Petri net (PN) is a 5-tuple \((P,T,PRE,POST,M0)\), where
P is a set of places \{p_1, p_2, \ldots, p_m\}, m \geq 0,
T is a set of transitions \{t_1, t_2, \ldots, t_n\}, n \geq 0,
where \( P \cap T = \text{null} \),
PRE : \( P \times T \rightarrow \mathbb{N} \) is the forward incidence function
defined on the set of arcs leading from
places to transitions,
where \( \mathbb{N} = \{0, 1, 2, \ldots\} \),
POST : \( P \times T \rightarrow \mathbb{N} \) is the backward incidence function
defined on the set of arcs leading from
transitions to places,
M_0 : \( P \rightarrow \mathbb{N} \) is the initial marking (see Definition 2-2).

Notes on Definition 2-1

In particular, for basic Petri nets, \( \mathbb{N} = \{0, 1\} \). Thus,
we have:

\[
\text{PRE}(p, t) = \begin{cases} 
1 & \text{if there is an arc from place } p \text{ to } \\
& \text{transition } t \\
0 & \text{otherwise }.
\end{cases}
\]

\[
\text{POST}(p, t) = \begin{cases} 
1 & \text{if there is an arc from transition } \\
& \text{transition } t \text{ to place } p \\
0 & \text{otherwise }.
\end{cases}
\]

Example 2-1

In Figure 2-1,

\[
\text{PRE}(1, 1) = 1, \quad \text{PRE}(1, 2) = 0, \\
\text{POST}(1, 1) = 0, \quad \text{POST}(1, 2) = 1.
\]
Definition 2-2

A marking $M$ is an $m$-vector denoting a distribution of tokens in the places. Notationally, $M = (M(p_1), M(p_2), \ldots, M(p_m))$, where $M(p_i)$ is the number of tokens residing in place $p_i$.

An initial marking $M_0$ is the marking of the Petri net at an initial moment.

Example 2-2

In Figure 2-1, initial marking $M_0 = (1,1,0,0)$.

Definition 2-3

The set

$\cdot t = \{ p \mid p \in P \text{ and PRE}(p,t) \neq 0 \}$

(resp., $\cdot t. = \{ p \mid p \in P \text{ and POST}(p,t) \neq 0 \}$)

is called the set of input (resp., output) places of transition $t$.

The set

$\cdot p. = \{ t \mid t \in T \text{ and POST}(p,t) \neq 0 \}$

(resp., $\cdot p. = \{ t \mid t \in T \text{ and PRE}(p,t) \neq 0 \}$)

is called the set of input (resp., output) transitions of place $p$. 

- 8 -
Example 2-3

In Figure 2-1,
\[ t_1 = \{ p_1, p_2 \}, \quad t_2 = \{ p_3, p_4 \}, \]
\[ p_1 = \{ t_2 \}, \quad p_2 = \{ t_1 \}. \]

Definition 2-4

The incidence matrix of a PN is an mxn matrix

\[ C = \begin{pmatrix} c_{ij} \end{pmatrix} \]

where \( c_{ij} = \text{POST}(p_i, t_j) - \text{PRE}(p_i, t_j). \)

Example 2-4

For Figure 2-1, the incidence matrix is

\[
C = B - A = \begin{bmatrix}
-1 & 1 & 0 \\
-1 & 0 & 1 \\
1 & -1 & 0 \\
1 & 0 & -1
\end{bmatrix}
\]

where \[ A = \begin{pmatrix} \text{PRE}(p_i, t_j) \end{pmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \]

and \[ B = \begin{pmatrix} \text{POST}(p_i, t_j) \end{pmatrix} = \begin{bmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
1 & 0 & 0
\end{bmatrix} \]
Definition 2-5

A transition \( t \) is said to be enabled for a marking \( M \), if
\[ M(p) \geq \text{PRE}(p,t), \quad \forall p \in .t \ . \]

Example 2-5

For Figure 2-1, \( M_0 = (M_0(p_1), M_0(p_2), M_0(p_3), M_0(p_4)) = (1,1,0,0) \), \( \text{PRE}(p_1,t_1) = \text{PRE}(p_2,t_1) = 1 \). Thus, \( t_1 \) is enabled for marking \( M_0 \).

2.1.2 Execution of PN

A Petri net is executed by firing one of its transitions. A transition may be fired if it is enabled. The firing of transition \( t \) for a marking \( M \) leads to a new marking \( M' \). The operation can be mathematically expressed as
\[ M ( t > M' ) \]
where
\[ M'(p) = M(p) - \text{PRE}(p,t) + \text{POST}(p,t), \quad \forall p \in P. \quad (1) \]

Equation (1) is called the firing equation for transition \( t \).

Example 2-6

In Figure 2-1, if \( M_0 = (1,1,0,0) \), then \( t_1 \) is enabled. The new marking obtained by firing \( t_1 \) is \( M' = (0,0,1,1) \).
There are two simple rules concerning the firing of transitions:

i) Only one transition of the PN may fire at a time, and the whole firing process of a transition is indivisible. This means that the process of firing a transition cannot be interrupted. Therefore, we can assume that the firing of a transition occurs at a single instant.

ii) If, at one moment, more than one transition is enabled, the transition selected for firing is not specified in the PN itself. The selection is usually determined by the application under consideration.

Suppose a sequence $\sigma$ of firing transitions leads a PN from marking $M$ to $M'$. The new marking $M'$ is given by the following matrix equation:

$$M' = M + C \cdot \bar{\sigma}$$

where $C$ is the incidence matrix of the PN,

$\bar{\sigma} = (\bar{\sigma}_1, \bar{\sigma}_2, \ldots, \bar{\sigma}_n)$ is the firing vector of the transition sequence $\sigma = (t_{j_1}, t_{j_2}, \ldots, t_{j_k})$,

with $1 \leq j_i \leq n$, $1 \leq i \leq k$, i.e., $\bar{\sigma}_i$ is the number of occurrences of $t_i$ in sequence $\sigma$.

Execution halts when no enabled transitions exist. The reachability set of a marking $M$, denoted by $[M]$, is the set of markings each obtained from marking $M$ by firing a transition sequence $\sigma$. 
\[
[M] = \{M^\sigma | M \sigma M^\sigma, \sigma \in T^\omega\}
\]

where \(T^\omega\) is the Kleene closure of \(T\).

The reachability set of the initial marking \(M_0\), denoted as \([M_0]\), is called the \textit{reachability tree} of the PN.

**Example 2-7**

In Figure 2-1, for \(M_0 = (1,1,0,0)\), the reachability tree is shown in Figure 2-2.

![Reachability Tree Diagram](image)

Figure 2-2: The Reachability Tree of the PN in Figure 2-1

### 2.2 EXTENSIONS OF PETRI NETS

Basic Petri nets are quite limited in their capabilities in modeling for real-life systems. Limitations include the following restrictions:

i) Only one type of token can be used.

ii) It is complicated to express certain constraints on events.
iii) It has no facility for delineating time-related
factors of a real-life system.

Several proposals have been put forth for increasing
their modeling power, such as PN with constraints, multiple
arcs, exclusive-OR transitions, switches, inhibitor arcs,
priorities, token colours and times [PETE77] [PETE81]. In
the following subsections, we describe some of the better
developed models, as illustrations of the diversity of petri
net's extended modeling capabilities. They include:

a. Token-coloured PN;
b. Predicate/Action PN;
c. Timed PN;
and

d. PN with Inhibitor Arcs.

More types of models may be obtained by combining some of
these types. Numerical PN (NPN), for instance, can be taken
as a combination of token-coloured PN and predicate/action
PN. NPN and token-coloured predicate/action PN are
equivalent [DAIZ82].

\[ \text{2.2.1 Token-coloured PN} \]

In a token-coloured PN, tokens are allowed to have
different natures (represented in terms of colours) and
values.
Example 2-8 [JENS80]

As an example, we consider the dining philosophers problem (for 5 philosophers). The philosophers alternately think and eat. To eat, a philosopher needs two forks, but unfortunately there are only five forks on a circular table and each philosopher is allowed to use only the two forks one on each of his sides. Obviously two neighbours cannot eat at the same time.

Figure 2-3 shows a predicate/action PN description for this problem. It uses 10 token-colours to distinguish the 5 philosophers and the 5 forks. In this figure, \( i \oplus 1 \) means \((i+1) \mod 5\). If a philosopher \( p_i \) is in the place 'THINK' and his two forks \( f_i \) and \( f_{i+1} \) are in the place 'FREE FORKS', then he can eat, i.e., transition \( t_l \) can be fired.

The marking \( M \) of a token-coloured PN has become a matrix:

\[
M = ( M(p,k) ) ,
\]

where \( M(p,k) \) is the number of tokens with the kth colour in place \( p \).

The forward and backward incidence functions have three independent variables:

\[
\text{PRE}(p,t,k), \quad \text{POST}(p,t,k),
\]

where \( k \) denotes the kth colour of the tokens.
Figure 2-3: A Token-Coloured PN for the Dining Philosophers Problem

A transition \( t \) is thus enabled under a more complicated condition:

\[
M(p,k) >= \text{PRE}(p,t,k), \forall p \in P \text{ and } \forall k.
\]

This type of PN is also known as place-coloured PN [DAI82]. It has been used for modeling different types of messages in information flow [BILL82].

2.2.2 Predicate/Action PN

In this type of PN [DAI82], predicates are associated with transitions and their input places, as additional conditions for the fireability of the transitions; and actions are associated with transitions, as operations to be done after firing the transitions. The predicates and
actions are expressions with numeric and/or logical variables describing the system.

The markings, arcs, and incidence matrices for a predicate/action PN are the same as for a basic PN. However, the fireability conditions and firing operations of a transition are changed from what are in basic PN to the following.

The conditions for firing a transition t include:

i) \[ M(p) \geq \text{PRE}(p,t) \], \( \forall p \in P \), and
ii) all the predicates associated with t and \( \forall p \in t \) should be true.

On firing transition t, the following operations are executed:

i) \[ M'(p) = M(p) - \text{PRE}(p,t) + \text{POST}(p,t) \].

ii) All the actions associated with t are carried out.

Example 2-9

Figure 2-4 is a predicate/action PN taken from [BILL82], in which t1 is attached with predicate \([Z=K]\) and t2 with action \(Y := Y + 1\). On firing t2, not only p3 and p4 each get a token, the value of variable \(Y\) is also increased by 1. At this moment, the value of variable \(Z\) should be equal to constant \(K\) in order to fire t1.
2.2.3 Timed Petri Nets

There are two different approaches to extending basic PN to this particular type. In the first approach, called Timed PN, a non-negative rational number is attached to each transition in the PN to denote its firing duration. In [MAG084], it is used for evaluating the cycle time of a PN within which the PN changes its state from the initial marking M0 to M0 again by firing a transition sequence.

In the second approach [DAI282] [MERL76a], called Time (out) PN, each transition is associated with a pair of values \([t^*, t^{**}]\), where \(t^*\) is the minimal time that the transition, after being enabled, must wait before it can be fired; \(t^{**}\) is the maximal time before which the transition must be fired, if it is still enabled. This approach is useful for modeling recoverable systems [MERL76a]. That is, there are some transitions used for recovering the loss of
tokens (representing lost messages in information flow). We will explain the application of TPN for specifying the timeout mechanism of communication protocol systems in Chapter 3.

2.2.4 Petri Nets with Inhibitor Arcs

A Petri net with inhibitor arcs is another major extension from basic PN [PETE81] [SCHW82]. An inhibitor arc is denoted by an arc with a small circle at its arrowhead [PETE81], such as the arc from p5 to t4 in Figure 2-5. It presents the condition that transition t4 cannot be enabled unless \( M(p5) = 0 \).

![Diagram of a Petri net with inhibitor arc]

Figure 2-5: A PN with An Inhibitor Arc

Some other extensions of PN, including the introduction of priorities among transitions, time bounds on transition
firings, or constraint sets that prohibit tokens from residing simultaneously in more than one place, are equivalent to PN with inhibitor arcs [PETE77].

A PN with inhibitor arcs can be converted into a basic PN with only general arcs [MOAL78]. Petri nets with inhibitor arcs, therefore, can be analyzed by using the same techniques as for basic PN and applying our package PNPUMO (see next section and Chapter 3).

2.3 **ANALYSIS BY PETRI NETS**

This section discusses how to analyze a general system by applying the techniques of PN. The aim is to determine the presence or absence of desirable or undesirable properties. This provides important insights into the behavior of a system.

2.3.1 **Problems for Analysis**

Before presenting the techniques for analysis, we first classify the properties of a PN according to three aspects: token-related properties, execution-related properties, and structural properties.

(1) **Token-related properties** include safeness, boundedness, and token conservation [PETE81]. They are defined below.
Definition 2-6

A PN is safe if the number of tokens in each place never exceeds 1. That is,

\[ M(p) \leq 1, \forall p \in P, \forall M \in [M_0] . \]

Definition 2-7

A PN is bounded if there exists an integer \( K \geq 0 \) which the number of tokens in each place never exceeds. That is,

\[ M(p) \leq K, \forall p \in P, \forall M \in [M_0] . \]

Definition 2-8

A PN is conservative if the total number of tokens in the net remains constant, that is

\[ \sum_{p \in P} M(p) = \sum_{p \in P} M_0(p), \forall M \in [M_0] . \]

(2) Execution-related properties include liveness and reachability. A PN is live if for any marking \( M \in [M_0] \) and any transition \( t \in T \), there always exists at least one sequence of transitions including \( t \) can be fired from \( M \). The liveness of a PN concerns the following status of its markings:

Deadlock - In this status, no transition can be fired.
Livelock - In this status, marking M is included in an execution loop without any transition leaving the loop.

Live-looping - In this status, marking M is included in an execution loop which has at least one possible transition leaving the loop.

Home-state - In this status, Marking M is the same as the initial marking, i.e., M = M₀, but is not the root of the reachability tree.

Another important concern in analysis is the reachability problem [PETE77]: Given a PN and a marking M, is M reachable from M₀? An algorithm and an excellent review for the general Petri net reachability problem appears recently in [MAYR84].

(3) Structural Properties

The structure of a PN is conveyed through place invariants and transition invariants of the net. A place invariant states that, during execution of a PN, the total number of tokens within a specific subset of places remains constant. A transition invariant represents a sequence of firing transitions which have no effects on a marking M [BERT82], i.e., M (σ > M).
2.3.2 Techniques for Analysis

There are two major techniques for analyzing systems described by Petri nets: enumerative analysis and structural analysis [JURG84]. The first technique is based on executing the PN and generating its reachability tree. The second is related to the structure of the net and the existence of place and transition invariants. More explanation is given below.

Enumerative Analysis

The reachability tree of a PN has its root at the initial marking M0. Its nodes represent the reachable markings and its arcs are labelled with the transitions fired.

The reachability set of a PN may be finite or infinite. To reduce an infinite reachable set to a finite representation, the symbol \( \omega \) is used to map many markings into one node of the reachability tree. That is, if the number of tokens in a place becomes arbitrarily large during execution of the PN, then we use \( \omega \) to represent the number of tokens in that place [PETE77] [PETE81]. Section 4.4 includes an example of a PN whose reachability set uses the symbol \( \omega \).

The procedure of enumerative analysis begins with generating the reachability tree. The generated markings (i.e., nodes of the tree) are checked. If a marking is in
the status of deadlock, or livelock, or live looping, or home-state, it is set as a terminal node, a leaf of the tree. Other nodes are called internal nodes.

The reachability tree provides an overview of the Petri net states (markings). From its nodes, all the token-related properties (safeness, boundedness, conservation) are easily detected. Liveness of the net is also observable from its leaves (possibly a deadlock, livelock, etc.).

**Structural Analysis**

In structural analysis, we get two sets of invariants (if they exist) - place invariants and transition invariants. Interpretation of these invariants is problem-dependent. Examples for protocol verification are given in Chapter 5.

(1) Place invariants are the solutions of the matrix equation

\[ \Phi \cdot C = 0, \]

where \( \Phi \) is an \( m \)-vector (\( m \) is the total number of places of the PN) and \( C \) is the incidence matrix of the PN. Suppose \( R \) is a solution of this equation, and for any new marking \( M' \) obtained by firing a sequence of transitions \( \sigma \) from a marking \( M \), we have \( M' = M + C \cdot \sigma \). Then by left multiplying both sides by \( R \), we get

\[ R \cdot M' = R \cdot M + R \cdot C \cdot \sigma = R \cdot M. \]
This equation implies that, if the values of the components of \( R \) are 1 or 0, the total number of tokens in the places associated with the non-zero components of \( R \) remains constant. Thus, we can say that a place invariant corresponds to a set of places where the total number of tokens is constant.

(2) Transition invariants are the solutions of the equation \( C \cdot \Omega = 0 \). A transition invariant is related to a transition sequence which has no effects on the marking. Let \( S \) be an \( n \)-vector solution to this equation. Suppose \( \sigma' \) is the sequence of transitions corresponding to \( S \), then \( \overrightarrow{S} = S \). For the new marking \( M' \) obtained by firing \( \sigma' \) from a marking \( M \), we have \( M' = M + C \cdot \overrightarrow{\sigma} \). Since \( C \cdot \overrightarrow{\sigma} = C \cdot S \), we get \( M' = M \). Therefore, firing the transition sequence \( \sigma' \) has no effects on any marking. This means that \( S \) is a transition invariant of the PN.

Thus, structural analysis involves essentially solving two linear algebraic equations \( \Phi \cdot C = 0 \) and \( C \cdot \Omega = 0 \).
Chapter III

PROTOCOL SPECIFICATION AND VERIFICATION BY PETRI NETS

Communication network protocols are extremely complicated distributed systems. They are prone to a lot of design errors, such as deadlocks, unspecified receptions, etc. It is very useful to have some formal methods for their modeling and verification so that they can be clearly specified and their design errors can be detected before implementation.

Because of their powerfulness for mathematical manipulation and for describing concurrency of systems, Petri nets, including their extensions, have been found as suitable tools for modeling and analysing the behaviours of both the control aspect (connection and disconnection) and the data transfer aspect of communication protocols.

This chapter reviews briefly the methods of applying PN to the specification of protocols and verification of their logical correctness. Though one of the main themes of this thesis is protocol verification, we consider also models for specification, as methods for verification are based on these models. The review provides the background knowledge for our applying PNPUO, an interactive Petri net package we
developed at the University of Ottawa, to the verification of protocols. Section 3.1 addresses the basic concepts of protocol specification and verification. Section 3.2 explains how to use various PN models in the specification of protocol entities, virtual media and error recovery mechanisms. Section 3.3 discusses the PN models for service specification. In Section 3.4, enumerative and structural analysis techniques for protocol verification are explained.

3.1 SPECIFICATION AND VERIFICATION OF PROTOCOLS

This section presents the concepts of protocol specification and verification.

According to the OSI Reference Model, each communicating system is organized as a hierarchy of up to seven layers. A protocol is a set of rules for governing the orderly exchange of control and data messages among a number of peer entities residing in these communicating systems [ISO81a] [TANE81].

In fact, messages are not directly transferred between entities of the same layer (except at the lowest layer). Instead, a message at layer N+1 is passed on to the lower layer by using the service primitives provided by layer N (i.e., operations at the interface between layer N+1 and layer N). The set of primitives provided by layer N to its user at layer N+1 and their way of operations forms the layer N service. Figure 3-1 illustrates this concept.
A service specification defines these service primitives provided by layer N. It describes only those operations executable by the user at layer N+1.

A protocol specification describes the interactions between peer layer N entities via the layer (N-1) service. According to Courier [COUR84], it should include the following three aspects:

a. Specification of the Communicating Protocol Entities
   This describes the specific interactions between the entities at the same layer [COUR84] [DAI82] [SID82].

b. Specification of the Communicating Virtual Medium
   This is required because entity interactions more or less depend on the behaviour of virtual medium.

c. Specification of the Recovery Mechanism
Protocol systems use recovery mechanisms such as
time-out to recover from failures.

There exist many modeling tools for specifying protocols,
such as finite state machines, Petri nets, abstract data
types, high level languages, temporal logic, etc. [DAIZ82]
[DANT80]. In Section 3.2, we shall describe some well-known
Petri net models for this purpose.

The objective of verifying a protocol is mainly to
demonstrate its logical correctness and its conformance with
the service specification. Logical correctness of a protocol
includes the following aspects [PETR] [SIDH82]:

Freedom from Deadlocks - The entities never get into
a state in which no more message transmissions
or receptions are possible.

Livelock Freeness - The entities never get into
any non-progressive loop of states.

Correct Termination - If started from the initial
state of the protocol, the entities always reach
the initial state again.

Freedom from Channel Overflow - The total number of
messages in each channel does not exceed a
fixed number (channel capacity).

If a protocol conforms with the service specification, it
should provide the services to the upper layer entities in
the specified manner.
3.2 Protocol Specification Using PN Models

This section describes several existing PN models for the specification of protocol components: protocol entities, virtual media, error recovery mechanisms, etc.

3.2.1 Specification of Protocol Entities by PN

Depending on its functions, a protocol entity may be considered as a Sender or as a Receiver. Three major types of PN models have been used for their specification: basic PN models, predicate/action PN models, and Timed PN models.

1) Basic PN Models (BPN)

Figure 2-1 can also be viewed as a basic PN model for a pair of Sender and Receiver. The left part (pl, p3, t1, and t2) describes the Sender, the right part the Receiver. Here, t1 indicates 'sending a message'; t2 represents 'get the data'; and t3 is for 'consume the data'.

Basic Petri net (BPN) models have been used for specifying several protocol systems recently. Tanenbaum [TANE81] presented a BPN model for the alternating bit protocol at the data link layer. Berthelot and Terrat [BERT82] introduced a BPN model for the transport layer entities (Connection and Disconnection Phases). These two models will be used for analysing protocols in Chapter 4. They will be described in details later.
It is intuitive and convenient to verify protocols, specified by BPN models, using the general analysis methods of Section 2.3. However, if a protocol is complicated, its BPN model will be considerably large and tedious to handle. The situation will be improved if we employ some predicates and actions in the model to represent constraints and events.

(2) Predicate/Action PN Models

Predicate/action PN models can be used for modeling local protocol entities. As a special form of predicates and actions, specific notation is used by Ayache [AYAC82] and Courtiat [COUR84] for exchanging messages:

\[ p?m \] means reception of message \( m \) sent by entity \( p \);

\[ p!m \] means transmission of message \( m \) to entity \( p \).

\( p!m \) is associated to the transitions of the entity 'Sender' as an action 'sending a message', \( p?m \) to the transitions of the 'Receiver' as a predicate 'receiving a message'. Their CAD protocol verification system OGIVE works on these types of predicates and actions only.

By using predicates and actions, an extended PN becomes a better model for complicated protocols. Methods for analysis, however, are not so simple, as they cannot be extended directly from the methodology of basic PN. More elements have to be taken into account during the process of
reachability analysis: all the predicates and actions, the values of all the variables which are involved in the expressions of the predicates or actions (see Section 2.2).

(3) Timed PN Models

Both the basic PN model and predicate/action PN model do not describe the time-related properties of protocols, such as time-out, time delay, and cycle time.

Obviously Timed PN models have advantages in this respect, since they have included the firing time period of every transition in the Petri net. It is easy to calculate the time delay and to realize the function of timeout mechanism. Many research papers are devoted to the application of timed PN to protocol specification and analysis [MAGO84] [MENA83] [MERL76a]. A typical TPN model for an alternating bit protocol is introduced by Menasche [MENA83]. In this model, as shown in Figure 3-2, places p1, p2, p3, p4 and transitions t1, t2, t3, t4, t5, t6 on the left form the Sender, those on the right form the Receiver, and the central part is the channel. Each transition is associated with a time duration. For example, the time duration [5,6] is associated with the timeout transition t2, meaning that retransmission of a message must take place in a time interval between 5 and 6 units after the last copy of the message has been sent.
3.2.2 Specification of Virtual Media by PN

In a specification of media, we may make different assumptions on the virtual medium: perfect medium, messages may be lost, messages may be duplicated, FIFO, bounded capacity of the channels, etc. There exist several methods for specifying the medium, such as direct coupling, virtual channel and distantly initiated action. In the following, we describe these methods in terms of Petri net models.

(1) Direct Coupling

Direct coupling is a technique in which a pair of labels !m and ?m are attached with two transitions, one in the Sender and the other in the Receiver, as an indication of the transfer of message m. We say that these two transitions are coupled, meaning that no explicit component in the model
is used to describe the virtual medium. The event 'message loss' can be specified by labelling a transition with \( \text{Im} \) in Sender without coupling with any transition in the Receiver \[\text{AYAC82}] \[\text{COUR84}].

(2) Virtual Channels

This approach is based on explicit modeling of any message in transit. The behaviours of the communication medium mentioned at the beginning of this subsection are specified as follows.

a. Perfect Medium and Medium Allowing Loss of Messages

If a medium is assumed to be perfect, there are no transmission errors. This is specified by using a set of places shared by two peer entities. As shown in Figure 3-3a, for each transfer direction, each place is related to a certain type of messages. A token inside a place indicates the existence of the corresponding message.

If loss of messages is possible, it is required to add and connect a transition without output places, called 'loss' in Figure 3-3b, to each of the shared places.
Figure 3-3: (a) A Perfect Channel (b) A Channel Allowing Message Loss

b. Bounded Capacity

Courtiat [COUR84] introduced a model for describing the message flow control aspect of a channel with limited capacity. As shown in Figure 3-4a, the portion with dotted arcs indicates the flow control mechanism. The number of tokens in place UE represents the maximum number of one type of messages allowed in the medium.

A model for several (say 2) types of messages is illustrated in Figure 3-4b.

c. FIFO virtual Channels

Courtiat [COUR84] also presented a model for the virtual channel with FIFO ordering of messages. As shown in Figure 3-5a, each slot of the queue has the same structure: place ES, place E, transition SE and transition 'Loss'. A token
inside place ES indicates that the jth slot of the queue is empty; a token inside E indicates that the jth slot of the queue contains a message of type E; transition SE denotes the shift of a message from slot j to slot j+1; transition 'Loss' denotes the possible loss of messages in the queue.

A model for several, say 2, types of messages is illustrated in Figure 3-5b. This model shows that two different types of messages can be transmitted through the same queue.

Another FIFO queue model using a predicate/action PN was introduced by Michel Daiz [DAIZ82]. In this model, a file F is used to carry the messages being transmitted. The 'sending MSG' transition in Sender is associated with an action 'DO APPEND(i,F)', which means that the message is to
be appended to \( P \). The receiving transition in Receiver is associated with another action \( \text{DO } j=\text{TAKEFIRST}(F) \), which indicates that the first message from \( F \) is removed to guarantee the FIFO feature.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fifomodel}
\caption{A FIFO Channel Model (a) For One MSG Type (b) For Two MSG Types}
\end{figure}
(3) Distantly Initiated Actions

Another technique for modeling the virtual medium was proposed by Bochmann [BOCH77]. In his model each communicating entity has certain distantly initiated actions which will be executed after a finite period of time following their initiation by the distant entity. These actions assign new values to local variables.

As an example, consider the alternating bit protocol. Sender is associated with a distantly initiated action TRANSA(p:(0,1)) to indicate the transmission of Acknowledgement, where parameter p is the MSG sequence number (0 or 1). Receiver has TRANSD(pl:(0,1); p2:...) to imply the transmission of data, where pl is the MSG sequence number, p2 is the data. In Sender, an action INITIATE(TRANSD,seq,data) is associated with the transition 'Emit'. In Receiver, an action INITIATE(TRANSA,seq) is associated with transition 'Assume Data', as shown below.

<table>
<thead>
<tr>
<th>Action attached with</th>
<th>Distantly Initiated Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender INITIATE(TRANSD,seq,data)</td>
<td>TRANSA(p:(0,1))</td>
</tr>
<tr>
<td>Receiver INITIATE(TRANSA,seq)</td>
<td>TRANSD(pl:(0,1); P2)</td>
</tr>
</tbody>
</table>

- 37 -
Once the 'Emit' transition of Sender is fired, the related action INITIATE(TRANS,seq, data) is executed, i.e. the distantly initiated action TRANS in Receiver is initiated. After a certain period of time, TRANS will be executed automatically and will assign new values to the local variables of Receiver: seqnb:=p1, data:=p2. This indicates the Receiver has received the message with the sequence number of pl which is the parameter 'seq' of the action with transition 'Emit' in the Sender.

Similar process occurs to TRANS. When Receiver processes a message (by firing transition 'Assume Data'), its action INITIATE(TRANS,seq) is executed. Thus, the distantly initiated action TRANS is initiated. Some time after TRANS has been executed, the Sender 'reaches' ACK and a communication cycle is fulfilled.

3.2.3 Specification of Recovery Mechanisms

Protocol systems ought to be recoverable from two cases of transmission anomalies:

Case 1. In this case, there are message errors (MSG misordered, duplicated, or distorted).

Case 2. In this case, messages are lost.

In both cases, retransmission of the message is required.
Message errors can be detected by checking sequence numbers and recovered by adding some transitions with logical predicates in the Sender and Receiver (see Figure 5 in [PETR] for details). Recovery from message loss is usually done by a time-out mechanism in the Sender. Time-out mechanisms can be implemented in BPN models, NPN models, predicate/action PN models or timed PN models. In most of the available PN models [MENA83] [MERL76a] [TANE81], a time-out mechanism is simply denoted as a transition called "timeout", as t5 shown in Figure 3-6a.

In the literature of these proposed models, except TPN, there is not much explanation on how to carry out the function "time is out". In the following, we describe our suggestions for how to get this timeout mechanism to work in each type of PN models.

\[\text{Figure 3-6: Time-out Mechanism (a) General Model (b) BPN Model}\]
In a NPN or predicate/action model, t5 can be associated with a predicate, which has variables common with the action associated with transition 'loss'. For instance, an action A:=1 may be attached with 'loss' and a predicate [A=1] attached to t5. If a message (token) is lost, i.e., 'loss' is fired, the action A:=1 sets A to 1. Then, the predicate associated with t5 is satisfied (A=1) and t5 may be fired (depending on whether p2 has a token or not). This approach is used in the PN model for the first protocol investigated in Chapter 5.

In a TPN model, the function 'timeout' is carried out as follows: Associate a time interval [t5*, t5**] with transition t5 (in Figure 3-6a) such that t5* > t3** + t4**, t5** is finite. Here, [t3*, t3**] and [t4*, t4**] are associated with transitions t3 and t4, respectively. Suppose a message or an acknowledgement of a message is lost. Having waited longer than a period t5**, transition t5 will be fired and send the message again (putting another token in place p7).

Merlin [MERL76a] explained the development of TPN for protocol recovery mechanism in [MERL76b]. Menasche [MENA83] presented an explicit example of TPN for the alternating bit protocol (see Figure 3-2).

For BPN models, time-out can be handled by a set of places and transitions forming a timer mechanism. For
example, as shown in Figure 3-6b, p9, p10, p11 and t6, t7 (called timer transitions) are added between t1 and t5. The following special firing rule is assumed: if an original transition (t1-5, 'loss's) and a timer transition (t6, t7) are enabled at the same time, the original transition should be fired first and then the firing of the timer transition follows. This rule guarantees that after waiting for a certain 'period' of time without firing any original transitions, t5 will be fired. For instance, if the first 'loss' transition is fired (a message is lost), then the only enabled transitions are t6, then t7. Finally t5 is fired - time out, the MSG is retransmitted.

3.3 SERVICE SPECIFICATION BY PN MODELS

An N-service specification describes the functions performed at layer N in response to service requests from the upper layer N+1. Petri nets are widely used for modeling these functions. Among these works, Billington's contribution [BILL82] is worth special attention. In his specification of Transport Service using predicate/action Petri nets, two variables, A and B, are used to represent the states of the interfaces between the protocol entities in layer N and their users in layer N+1, respectively. The four possible states of each interface are: Idle, Outgoing (Incoming) Connection Pending, Data Transfer Transfer Ready, and Disconnection Pending. Based on these two variables,
some predicates and actions are attached with the transitions. For example, in the Connection Phase (see FIG.4 in [BILL82]), predicate \([A=0]\) and action \(A:=1\) are associated with transition \(ST1\). It indicates that, once user \(A\) issues a Connect Request while being in the Idle state \((A=0)\), the interface will enter the Connection Pending state \((A=1)\).

In order to distinguish the service primitives, different coloured tokens may be used. Billington has defined 16 coloured tokens to represent the primitives of Connect Request, Connect Indication, Connect Response, Connect Confirmation, etc.

Courtiat, Ayache, Algayres and Daiz also made significant contributions in protocol service specifications [COUR84] [DAIZ82]. Their models are based on the special forms of predicates and actions : ?m and !m.

3.4 **PROTOCOL VERIFICATION USING PETRI NETS**

Two steps are involved in the verification of protocol by Petri nets:

1. **Step 1.** Specify the protocol using a PN model.
2. **Step 2.** Use PN analysis techniques of Section 2.3 to verify the PN model.
In Section 3.2 we have already discussed the PN models for specification of protocol components. There are two objectives in Step 2: To verify the logical correctness of a protocol and its conformance with the service specification. In this thesis, we concentrate on just the verification of logical correctness. Conformance of a protocol with its service specification will only be briefly mentioned in Section 3.4.2.

3.4.1 Logical Correctness

As mentioned in Section 3.1, the logical correctness of a protocol includes deadlock freeness, livelock freeness, correct termination and freedom from channel overflow. They correspond to the following properties of a PN:

- freedom from deadlocks
- livelock freeness
- M0 is a home-state
- boundedness

Enumerative analysis as described in Section 2.3.2 can be used to verify these logical properties. The transition invariants produced in structural analysis are also useful for determining the termination property (home-state), while the place invariants are useful for investigating the flow control property (boundeness).
We have developed a CAD software package called PNPVO to be used for the verification of protocols specified with token-coloured and predicate/action PN models. It is possible to specify, modify, store the PN models, to analysis their logical properties by enumerating of the reachable markings and to obtain the place and transition invariants. (For operational details of PNPVO, see Chapter 4).

In the timed PN model, the size of the reachability tree may be smaller than that in the corresponding BPN model. The reason of this is that pairs of time values are considered so that some enabled transitions are ignored from firing [MERL76b]. The rest of the analysis process is the same as in BPN models. Menasche and Bochmann [MENA83] proposed a new analysis method based on their new concept 'TPN state classes'.

3.4.2 Conformance of Protocol with Service Specification

The next step in protocol verification is to prove its conformance with the service specification. It is to check whether the protocol uses the service from a lower layer and performs the service for the upper layer as specified in the service specifications. Currently, this is becoming an important research issue of protocol testing. An automated
testing system and some references are given by Ural and Probert [URAL84].

The conformance of protocol with the service specification is a very profound subject itself and is out of the scope of our thesis.
Chapter IV

PNPUO - AN INTERACTIVE PETRI NET PACKAGE AT UNIVERSITY OF OTTWA

This chapter describes in details both the user interface and the system aspect of an interactive Petri net package implemented at the University of Ottawa (hereafter abbreviated as PNPUO). A brief overview of PNPUO is given in the first section. User interface with the system is described in Section 4.2. Section 4.3 explains the organization and functions of its various components. By means of an example, typical sessions of conversation between the user and the system are illustrated in Section 4.4. At the end of the chapter, we propose some possible improvements on the system and directions for further research.

4.1 OBJECTIVES AND CAPABILITIES OF PNPUO

PNPUO provides the user with interactive facilities for analysing properties of systems which use basic, or token-coloured, or predicate/action Petri nets as design models. This is done mainly in two phases:

1. Phase I Creation and Storage of Data
A user can interactively input, display, modify, store and get hard copies of the following input data:

a. The numbers of places, transitions, and token colours of the Petri net.

b. Labels (numbers and colours) of arcs from places to transitions and from transitions to places.

c. User variables (integer, real, Boolean) used in the predicates and actions of the Petri nets.

d. Predicates and actions at places and transitions.

e. Initial marking and initial values of variables.

**Phase II System Analysis**

After data have been correctly input, the system enters the analysis phase, during which the user can investigate properties of the system by the following two processes (see Section 2.3 for theoretical background):

(i). In the process of enumerative analysis, it checks:

a. Token-related properties: safeness, boundedness and conservation of tokens.

b. Execution-related properties: deadlocks, livelocks, live-loops and home-state.

c. Reachability of states.
(ii). In the process of structural analysis, it determines:

a. Place invariants.

b. Transition invariants.

4.2 USER'S INTERFACE WITH SYSTEM

PNPUO is a menu-driven system. At each step it provides the user with a set of options and instructions. It then passes control to an internal process determined by the user's selection.

A conversational session starts with PNPUO displaying the following information:

**********************************************************************
* DO YOU WANT TO ENTER: *
* 1. PROCESS MODULE SPECIFICATION *
* 2. MODULE DISPLAY *
* 3. MODULE MODIFY *
* 4. MODULE ANALYSIS *
**********************************************************************
* ENTER YOUR SELECTION *
* HIT ANY DIGITS OTHER THAN 1, 2, 3, 4 TO GET OUT OF THIS PROCESS *

Selection of an option will start the control hierarchy of the user interface, as shown in Figure 4-1. The different parts of this hierarchy are explained in details in Subsections 4.2.2 to 4.2.5. Subsection 4.2.1 summarizes the limitation in the application of PNPUO.
1. Process Module Specification
2. Module Display
3. Module Modify
4. Module Analysis

1. Entity Declaration
2. Predicate-Action Definition
3. Initial Marking & Variables

1. Total Numbers M, N, L
2. Arcs P - T & T - P
3. Integer Variables
4. Real Variables
5. Boolean Variables

1. PT - Predicate on Transition
2. AT - Action at Transition
3. PP - Predicate on Place
4. AP - Action at Place

1. Initial Marking MO
2. Initial Values of Variables

1. Enumerative Analysis
2. Structural Analysis

1. Reachability Tree
2. Token Properties
3. Execution Properties

1. Place Invariants
2. Transition Invariants

Figure 4-1. Control Hierarchy of User Interface
4.2.1 Scope and Limitation of Application

Though the framework of PNPUO is designed in such a way that many types of Petri nets can be analysed, the current version has only the software for token-coloured and predicate/action PN, including basic PN as a special case. Other types of Petri nets, such as timed PN and PN with inhibitor arcs, are not included. (See Section 4.5 for discussion on extending the scope of applications.)

The following limitations on the selection of maximum values of the PN parameters and on the formats of predicates and actions are mainly due to implementational convenience and restrictions. Most of them can be removed easily, possibly at the expense of larger storage space or computational efficiency.

Limitations

a. The maximum numbers of places, transitions, and token colours are limited to 40, 40, and 9, respectively;
b. The maximum number of each type of user variables, namely integer, real and Boolean, is limited to 8;
c. Use of Symbols for integer, real and Boolean variables is restricted to A, B, ..., H; A1, B1, ..., H1; and A2, B2, ..., H2, respectively. (It will be shown later that, in the Display Phase, three tables are used to list the user variables and the corresponding program variables of each type.)
d. A predicate must be one of the following two logical
expressions:

i) \( X = R \)

ii) \( X = R \) and \( Y = S \)

where \( X, Y \) are program variables of any type (integer, real, or Boolean), and \( R, S \) are constants of the same type as \( X \) and \( Y \), respectively;

e. An action must be one of the following two logical expressions:

i) \( X := X' + R \)

ii) \( X := X' + R \) and \( Y := Y' + S \)

where \( X, Y, R, S \) are the same as above and \( X' (Y') \) is either \( X \) (\( Y \)) or null.

f. The maximum number of levels of the reachability tree is 15.

Note that user variables are the variables used in the predicates or actions of the PN. Program variables are those used in the program modules.

4.2.2 Creating Modules for Analysis

Once the system enters the module CREATE, the following information will be displayed:
** ** 
MODULE SPECIFICATION 
< ON CREATE >

**********************************************************************
* DO YOU WANT TO ENTER : 
* 1. ENTITY DECLARATION PHASE 
* 2. PREDICATE-ACTION DEFINITION PHASE 
* 3. INITIAL MARKING & VARIABLES PHASE 
**********************************************************************
* ENTER YOUR SELECTION 
** HIT ANY DIGITS OTHER THAN 1, 2, 3 
TO GET OUT OF MODULE SPECIFICATION SESSION 

These three phases (i.e., options) are as follows:

(1) **Entity Declaration Phase**

In this phase, a user enters the data about a protocol entity, following the instructions displayed by the system:

** ** 
ENTITY DECLARATION PHASE 
< ON CREATE >

**********************************************************************
* DO YOU WANT TO DECLARE : 
* 1. NUMBERS OF PLACES, TRANSITIONS, AND COLOUR TYPES 
* 2. ARCS P --> T & T --> P 
* 3. INTEGER VARIABLES 
* 4. REAL VARIABLES 
* 5. BOOLEAN VARIABLES 
**********************************************************************
* ENTER YOUR SELECTION 
** HIT ANY DIGITS OTHER THEN 1, 2, 3, 4, 5 
GET OUT OF ENTITY DECLARATION PHASE 

Each option triggers the display of one of the following sets of instructions.

**Option 1.** To enter number of places, transitions and colour types.
<table>
<thead>
<tr>
<th>Displayed Information</th>
<th>Range of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Total No. of Places</td>
<td>$0 \leq M \leq 40$</td>
</tr>
<tr>
<td>Enter Total No. of Transitions</td>
<td>$0 \leq N \leq 40$</td>
</tr>
<tr>
<td>Enter Total No. of Token-colours</td>
<td>$0 \leq L \leq 9$</td>
</tr>
</tbody>
</table>

Option 2. To define arcs $p \rightarrow t$ & $t \rightarrow p$.

<table>
<thead>
<tr>
<th>Displayed Information</th>
<th>Range of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_i$  $t_j$</td>
<td>any digits</td>
</tr>
<tr>
<td>$t_i$  $p_j$</td>
<td>any digits</td>
</tr>
</tbody>
</table>

Here, the user should input the number of arcs from place $p_i$ to transition $t_j$ behind `$p_i \rightarrow t_j$`, and input the number of arcs from transition $t_i$ to place $p_j$ behind `$t_i \rightarrow p_j$`.

Option 3. To define integer variables $A$, $B$, $\ldots$, $H$.

<table>
<thead>
<tr>
<th>Displayed Information</th>
<th>Range of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Total No. of Integer Var.</td>
<td>$0 \leq k \leq 8$</td>
</tr>
<tr>
<td>Enter name of $i$th Integer Var.</td>
<td>character string</td>
</tr>
<tr>
<td></td>
<td>$(\text{length} \leq 80)$</td>
</tr>
</tbody>
</table>
Option 4. To define real variables A1, B1, ...H1
(similar to Option 3).

Option 5. To define Boolean variables A2, B2, ...H2
(similar to Option 3).

(2) Predicate-Action Definition Phase

In this phase, the user inputs all the predicates and actions of the PN. The process starts with the following information displayed by the system:

```
** CONSTRANTS DEFINITION PHASE **
< ON CREATE >

******************************************************************************
* DO YOU WANT TO DEFINE : *
* 1. PT --- PREDICATES ON TRANSITIONS *
* 2. AT --- ACTIONS AT TRANSITIONS *
* 3. PP --- PREDICATES ON PLACES *
* 4. AP --- ACTIONS AT PLACES *

******************************************************************************
* ENTER YOUR SELECTION *
* PRESS ANY DIGITS OTHER THAN 1, 2, 3, 4*
* TO GET OUT OF CONSTRAINTS DEFINITION PHASE *
```

These options are explained below:

Option 1. To enter PT --- Predicates on Transitions.

<table>
<thead>
<tr>
<th>Displayed Information</th>
<th>Format of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ti</td>
<td>Character string as</td>
</tr>
</tbody>
</table>

```
X = R or X = R and Y = S
```
**Option 2.** To enter AT — Actions at Transitions.

<table>
<thead>
<tr>
<th>Displayed Information</th>
<th>Format of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ti : k</td>
<td>Character string as</td>
</tr>
<tr>
<td></td>
<td>$X := X' + R$ or</td>
</tr>
<tr>
<td></td>
<td>$X := X' + R$ and $Y := Y' + S$</td>
</tr>
</tbody>
</table>

**Option 3.** To enter PP — Predicates on Places (Similar to Option 1).

**Option 4.** To enter AP — Actions at Places (Similar to Option 2).

(3) **Marking and Variable Initialization Phase**

In this phase, the user enters the initial marking and the initial values of each type of variables. The interactive process is shown in the following table.
<table>
<thead>
<tr>
<th>Displayed Information</th>
<th>Range of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pi</td>
<td>any integer (value of $\pi$)</td>
</tr>
<tr>
<td>A (or B, ... H)</td>
<td>any integer (value for A or B, ... H)</td>
</tr>
<tr>
<td>A1 (or B1, ... H1)</td>
<td>any real number (value for A1 or B1, ... H1)</td>
</tr>
<tr>
<td>A2 (or B2, ... H2)</td>
<td>&quot;true&quot; or &quot;false&quot; (value for A2 or B2, ... H2)</td>
</tr>
</tbody>
</table>

### 4.2.3 Displaying of Data

By means of the DISPLAY module, the system allows two ways of displaying information by asking the user the following question:

**WHICH WAY OF DISPLAY DO YOU WANT TO CHOOSE:**

A. DISPLAY ALL THE INFORMATION AT ONCE  
B. DISPLAY INFORMATION INTERACTIVELY

If Option A is chosen, the system displays all of the input data together, i.e., including:

a) Total numbers of places, transitions, and token colours.

b) Arcs from places to transitions and arcs from transitions to places.

c) The list of user variables and their correspond-
ponding program variables.

d) Predicates and actions at places and transitions.
e) Initial marking and initial values of variables.

If Option B is chosen, the system displays these input data part by part. At each step, it displays only that part of data the user requests.

4.2.4 Modifying Modules

The process of modifying a module is similar to the process of module displaying as described in Subsection 4.2.3, except that, in the modification phase, each line of displayed data is followed by a question:

* Do you want to change? —— Y or N *

If the user presses 'Y', the system displays:

* Enter the new line *

The user then inputs the new data.

4.2.5 Analysing Modules

In the analysis phase, the system will enter the process of reachability analysis or incidence analysis according to the user's request. At first, the system displays the following information:
**MODULE ANALYSIS**

***********
* DO YOU WANT TO DO ?
  * 1. REACHABILITY ANALYSIS
  * 2. INCIDENCE ANALYSIS
***********

** ENTER YOUR SELECTION **
* HIT ANY DIGIT OTHER THEN 1 AND 2 TO END ANALYSIS *

The interactive process for each kind of analysis is described below:

(I) Enumerative Analysis

```
<table>
<thead>
<tr>
<th>Displayed Information</th>
<th>User's choice</th>
</tr>
</thead>
</table>

(if the PN has predicate/action: ) |
Do you want give variable values from |
terminal during analysing ? |
( if 'Y', after getting each node of |
the reachability tree: ) |
Enter new values (if any) to the va- |
variable(s) (e.g. A=1). If no entry 'N' or character |
p PRESS N |
| string as A=1.. |
```

The outputs of this process consist of:

i) the reachability tree with terminal labels on the leaves,

ii) token-related properties (safeness, boundedness, and conservation), and
iii) execution-related properties (deadlocks, live-
    (deadlocks, livelocks and live loops).

(II) Structural Analysis

No interactions are needed in this process. Results about
transition invariants and place invariants are automatically
output.

4.3 IMPLEMENTATIONAL DETAILS

PNPUO was coded in PASCAL 8000 and implemented in the
main frame computer AMDAHL at the University of Ottawa. It
operates under the Operating System CMS (Conversational
Monitor System). Its main program controls the operations of
two major components: one used for the construction of PN
and the other for the analysis of PN.

Module CONSTRUCTION consists of three components:

Module CREATE

Module DISPLAY

and

Module MODIFY.

Module ANALYSIS includes two parts:

Enumerative Analysis

and

Structural Analysis.
A brief overview of the program organization is shown in Figure 4-2. Details of the various parts are described in the following subsections.

![Diagram of program organization]

Figure 4-2: Program Organization of PNPUCO

4.3.1 Main Program

The main program serves three functions:

a. At the beginning, it opens all I/O files and initializes all program flags.

b. Next, it invokes the procedure NEW to display the first page of information:
c. Subsequently, it interacts with the user by invoking procedures in response to user's selection of options.

4.3.2 Module CREATE

The module CREATE stores the input data into arrays and assigns values to variables. This is realized in three phases: Entity Declaration, Input of Constraints, and Variable Initialization.

The procedures of Module CREATE and their parameters are summarised in TABLE 4-1.

Having received all the information, Module CREATE invokes the external procedure CWRITE to store the numerical data and character data into the files OUTIN DATA and OUT DATA, respectively. Before logoff, copies of these two files, DATAIN DATA and DIN DATA are produced and stored automatically for subsequent usage.
<table>
<thead>
<tr>
<th>Phase</th>
<th>Procedure</th>
<th>Input data</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDECLARATION</td>
<td>CNUMBER</td>
<td># of places, transitions &amp; colours</td>
<td>Var M, N, L</td>
</tr>
<tr>
<td>(entity</td>
<td>CARC</td>
<td>Arcs P-1T &amp; T-]P</td>
<td>Array PARC, TARC</td>
</tr>
<tr>
<td>declaration)</td>
<td>CINTEGER</td>
<td>Integer variables</td>
<td>Var A, B,...H</td>
</tr>
<tr>
<td></td>
<td>CREAL</td>
<td>Real variables</td>
<td>Var A1, B1,...H1</td>
</tr>
<tr>
<td></td>
<td>CLOGIC</td>
<td>Boolean variables</td>
<td>Var A2, B2,...H2</td>
</tr>
<tr>
<td></td>
<td>CPREDTR</td>
<td>Predicates on transitions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CACTTR</td>
<td>Actions at transitions</td>
<td>Array PT</td>
</tr>
<tr>
<td></td>
<td>CPREDPL</td>
<td>Predicates on places</td>
<td>Array AT</td>
</tr>
<tr>
<td></td>
<td>CACTPL</td>
<td>Actions on places</td>
<td>Array PP</td>
</tr>
<tr>
<td></td>
<td>CINITIAL</td>
<td>Initial marking M0</td>
<td>Array AP</td>
</tr>
<tr>
<td></td>
<td>CINIMARK</td>
<td>Initial variable values</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CINIVAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.3 Module DISPLAY

This module invokes three internal procedures:

Procedure DDECLARATION

Procedure DPREDACT

and Procedure DINITAL.

Each procedure has a similar set of subprocedures as Module CREATE. The difference is: Module CREATE prepares data whereas Module DISPLAY displays them.

To implement the two display Options A and B as mentioned in Subsection 4.2.3, a program flag, DFLAG, is set to the
value 0 or 1 according to the user's selection. Different parts of this procedure will then be executed for Option A or Option B.

TABLE 4-2 lists all the procedures and subprocedures of Module DISPLAY.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Sub-pro.</th>
<th>Function</th>
<th>Data Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDECLARATION</td>
<td>DNUMBER</td>
<td>display # of places, transitions &amp; colors</td>
<td>Var M, N L</td>
</tr>
<tr>
<td></td>
<td>DARC</td>
<td>display Arcs P -&gt; T &amp; T -&gt; P</td>
<td>Array PARC, TARC</td>
</tr>
<tr>
<td></td>
<td>DINTEGER</td>
<td>display integer Vars</td>
<td>Var A, B, ..H</td>
</tr>
<tr>
<td></td>
<td>DREAL</td>
<td>display real Vars</td>
<td>Var A1, B1, ..H1</td>
</tr>
<tr>
<td></td>
<td>DLOGIC</td>
<td>display Boolean Vars</td>
<td>Var A2, B2, ..H2</td>
</tr>
<tr>
<td>DPREDCANT</td>
<td>DPREDTR</td>
<td>display predicates on transitions</td>
<td>Array PT</td>
</tr>
<tr>
<td></td>
<td>DACTTR</td>
<td>display actions at transitions</td>
<td>Array AT</td>
</tr>
<tr>
<td></td>
<td>DPREDPL</td>
<td>display pred. on pl.</td>
<td>Array PP</td>
</tr>
<tr>
<td></td>
<td>DACTPL</td>
<td>display act. at pl.</td>
<td>Array AP</td>
</tr>
<tr>
<td>DINITIAL</td>
<td>DINMARK</td>
<td>display initial marking M0</td>
<td>Array P</td>
</tr>
<tr>
<td></td>
<td>DINIVAR</td>
<td>display initial values of Vars</td>
<td>Var A, ..H, A1, ..H1, A2, ..H2</td>
</tr>
<tr>
<td>DREAD</td>
<td>/</td>
<td>read numerical, character data</td>
<td>Files DATAIN DATA, DIN DATA</td>
</tr>
</tbody>
</table>
4.3.4 **Module MODIFY**

Before entering Module MODIFY, the main program sets the program flag MFLAG to 1 and then invokes procedure DISPLAY. In procedure DISPLAY, some specific program statements can only be executed when MFLAG = 1. These specific statements and the subprocedures MODI, DMOD, PMOD constitute the Module MODIFY.

At the end of this process, procedure CWRITE is invoked to store the modified data into the output files OUTIN DATA and OUT DATA.

4.3.5 **Module ANALYSIS**

This module contains two processes for enumerative analysis and structural analysis. They are implemented as procedures REACHABILITY and INCIDENCE, respectively.

I. Enumerative Analysis

This process delineates the reachability tree, token-related properties, and execution-related properties. (See Section 2.3)

During the generation of a reachability tree, the system determines whether a transition Tj0 is fireable or not by checking the following two conditions:
(1) Whether the transition $T_{j0}$ is enabled, i.e., whether

$$M(i,k) \geq PARC(i,j0,k)$$

$$\forall i \in \{i | P \in .T_{j0} \} \text{ and } \forall k \in \{K | 1 \leq k \leq 1 \},$$

(2) Whether all the predicates related to $T_{j0}$ are satisfied, i.e., whether

$$PT(j0,j) \ (1 \leq j \leq 81) \text{ is true, and}$$

$$PP(i,j) \ (1 \leq j \leq 81) \text{ is true}$$

$$\forall i \in \{i | P_i \in .T_{j0} \cup \{i | P_i \in T_{j0} \} \}.$$
or the tree level has exceeded 15 (maximum number of levels of the reachability tree allowed by the system.)

During execution of the Petri net, the system outputs the nodes of the reachability tree (with type-labels on its leaves). Then, the system summarizes the token-related properties, execution-related properties and liveness properties of the PN.

There are seven subprocedures in the process ANALYSIS:

1) PREP: It initializes all variables and arrays, and asks if the user wants to change variable values during the execution. If so, it sets flag GG=1.

2) RECEIVE: At each level, if GG=1, it receives values for the variables (like A=1) from the console.

3) CHECK: It checks token-related properties.

4) FIND: It finds fireable transitions. If Tj is fireable, it sets TTj=1. (Otherwise, TTj=0)

5) FIRING (J): It fires Tj, gets next marking Mk+l and executes the action associated with Tj and with the output places of Tj.

6) LEAF (J, UU): It tests whether Mk+l is a leaf or not. If so, it determines its type.

7) LOCK (J, VV): It detects deadlocks and livelocks.
The flowchart of Procedure REACHABILITY is shown in Figure 4-3.

II. Structural Analysis

In structural analysis, place invariants and transition invariants are determined by solving two linear equations:

\[ \Phi \cdot C = 0 \quad \text{and} \quad C \cdot \Omega = 0. \]

Procedure INCIDENCIA consists of subprocedures SOLUTION, FAI, and OMAGA. It obtains the incidence matrix C from arrays PARC and TAR\(C\). In fact, \( C = TAR\(C\) - PARC \). It then invokes Procedure OMAGA which, in turns calls Procedure SOLUTOIN to perform the elementary operations on matrix C. OMAGA then solves the equation \( C \cdot \Omega = 0 \) with only 0/1 solutions. It outputs sequences of transition invariants in the form.
Tj1; Tj2, ....... Tjk

M --------- -> M

The above indicates that the sequence of transitions Tj1 .... Tjk leads a marking M to M itself again. However, the process of INCIDENCE analysis does not point out which specific marking M is. It may or may not be the initial marking M0. To determine this, the user has to refer to the results of reachability analysis.

At this stage, INCIDENCE invokes Subprocedure FAI, which transposes the incidence matrix C, D = C'. Then it passes D on to Subprocedure SOLUTION to perform the elementary operations. At the next step, FAI solves \( \Phi \cdot C = 0 \) (also with only 0/1 solutions) and outputs the place invariant equations:

\[
\Phi \cdot M = \Phi \cdot M_0
\]

i.e., if \( v \) is the total number of solutions of \( \Phi \cdot C = 0 \), then for each \( h: 1 \leq h \leq v \), the printout will be

\[
M(p1) + M(p2) + \ldots + M(pj) = M0(p1) + M0(p2) + \ldots + M0(pj)
\]

where \( \Phi h(ik) = 0 \quad k: 1 \leq k \leq j \).

In total, there are \( v \) outputs similar to the above expression.
4.4 AN EXAMPLE SHOWING HOW PNPVO OPERATES

In this section, we illustrate the various typical user-system interactions through an example. The simple basic Petri net used in this example is shown in Figure 4-4.

![Petri Net Diagram]

Figure 4-4: Petri Net for the Example

The user starts PNPVO by typing the command 'RUN'. The following instructions will appear on the screen:

************
* DO YOU WANT TO ENTER:
* 1. PROCESS MODULE SPECIFICATION
* 2. MODULE DISPLAY
* 3. MODULE MODIFY
* 4. MODULE ANALYSIS
************
* ENTER YOUR SELECTION *
* HIT ANY DIGITS OTHER THAN 1, 2, 3, 4 TO GET OUT OF THIS PROCESS *

The user should select Option 1, so as to input the data of the Petri net. Then the system displays a set of instructions:
** ** MODULE SPECIFICATION ** **

< ON CREATE >

******************************************************************************
* DO YOU WANT TO ENTER :
*    1. ENTITY DECLARATION PHASE
*    2. PREDICATE-ACTION DEFINITION PHASE
*    3. INITIAL MARKING & VARIABLES PHASE
******************************************************************************

* ENTER YOUR SELECTION
* HIT ANY DIGITS OTHER THAN 1, 2, 3
TO GET OUT OF MODULE SPECIFICATION SESSION *

The user inputs all the data after these instructions:

Numbers of places, transitions and colours

( 4, 3 and 1 ),

Arrows from places to transitions and from transitions
to places:

- PARC(1,1) = 1, PARC(1,2) = 0, PARC(1,3) = 0,

- TARC(1,1) = 1, TARC(1,2) = 0, TARC(1,3) = 0,

etc.

Initial marking M0 : ( 1,0,1,0 ).

Next, if the user chooses Option A, Module DISPLAY will
display all the input data as follows.

<table>
<thead>
<tr>
<th>TOTAL # OF PLACES</th>
<th>M= 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL # OF TRANSITIONS</td>
<td>N= 3</td>
</tr>
<tr>
<td>TOTAL # OF TOKEN TYPES</td>
<td>L= 1</td>
</tr>
</tbody>
</table>

** ** ARCS P --> T & T --> P ** **

<table>
<thead>
<tr>
<th>PLACE</th>
<th>TO TRANS.</th>
<th>COL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P01</td>
<td>T01</td>
<td>1</td>
</tr>
<tr>
<td>P02</td>
<td>T01</td>
<td>1</td>
</tr>
<tr>
<td>P03</td>
<td>T01</td>
<td>1</td>
</tr>
<tr>
<td>P03</td>
<td>T03</td>
<td>1</td>
</tr>
<tr>
<td>P04</td>
<td>T02</td>
<td>1</td>
</tr>
</tbody>
</table>

** ** INTEGRAL VARIABLES ** **

<table>
<thead>
<tr>
<th>PROGRAM'S</th>
<th>USER'S</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; NIL &gt;</td>
<td></td>
</tr>
</tbody>
</table>

** ** REAL VARIABLES ** **

<table>
<thead>
<tr>
<th>PROGRAM'S</th>
<th>USER'S</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; NIL &gt;</td>
<td></td>
</tr>
</tbody>
</table>
** LOGIC VARIABLES **

PROGRAM'S USER'S

< NIL >

** PREDICATES ON TRANSITIONS **

TRANS. PREDICATE

T01 T02 T03

** ACTIONS AT TRANSITIONS **

TRANS. ACTION

T01 T02 T03

** PREDICATES ON PLACES **

PLACE PREDICATE

P01 P02 P03 P04

** ACTIONS AT PLACES **

PLACE ACTION

P01 P02 P03 P04

** INITIAL MARKING M0 **

PLACE COL.1

< DISPLAY >

P01 1
P02 0
P03 1
P04 0

If errors are found in this list, they should be corrected by calling the process MODIFY.

Finally, the user enters the process ANALYSIS and obtains the following information:

Reachability Tree (see Figure 4-5):

Token-related Properties (output by the system):

THE RESULTS OF REACHABILITY ANALYSIS:

< 1> THE NET IS NOT SAFE.

< 2> THE MAX NUMBER OF TOKENS IN EACH PLACE >= (FOR COLOUR) 1

< 3> THE NET IS NOT CONSERVED.
Figure 4-5: Reachability Tree of the PN in Figure 4-4

Execution-related Properties (output by the system):

<4> EXECUTION-RELATED PROPERTIES:
THE NET HAS DEADLOCK.<S>.
NO LIVE-LOCKS.
THE NET HAS LIVE-LOOPING.<S>.
M0 IS NOT A HOME-STATE.

Place Invariants (output by the system):

PLACE INVARIANT l:
M(l) = 1
%

Transition Invariants (output by the system):

* THERE IS NOT ANY TRANSITION INVARIANT FOR THIS NET *

It can be checked manually that the Petri net does have all the above properties. The fact that Place 1 always keeps one token is shown by the above place invariant.
Chapter V

PROTOCOL VERIFICATION BY PNPUO

This chapter describes the application of PNPUO (described in Chapter 4) in the field of communication network protocol verification. The theory and techniques behind these applications are developed in Chapters 2 and 3. For illustration purpose, we consider in details two protocol systems. Section 5.1 considers the alternating bit protocol, a well known protocol in the data link layer. The Connection and Disconnection Phases of Class 1 and Class 2 of the Transport Protocol are verified in Section 5.2, using both BPN and predicate/action PN. This protocol is of current research interest.

As mentioned in Chapter 3, only correctness of the logical properties of the protocols will be dealt with.

5.1 ANALYSIS OF THE ALTERNATING BIT PROTOCOL BY PNPUO

We assume a noisy channel (i.e., messages may be damaged or lost in the channel) for the alternating bit protocol. In this protocol, every message has a one-bit sequence number as an identifier. During transmission, the sequence number alternates between the values 0 and 1. A protocol
entity does not send the next message until the one sent previously has been positively acknowledged. Recovery from transmission anomalies (errors or loss) is achieved by a time-out mechanism and retransmission of messages.

A basic PN model with a recovery mechanism for this protocol, including a pseudo-code program, is described in [TANE81]. It does not explain how the time-out mechanism works. Based on this model, we develop a predicate/action PN model, in which the predicates and actions take care of the time-out mechanism.

3.1.1 A Predicate/Action PN Model

As shown in Figure 5-1, our predicate/action PN model for the alternating bit protocol uses two variables, A and B, to specify the functions 'message loss' and 'time-out'. It attaches predicate [A=1] and actions 'A:=0 and B:=1' to the transitions 'loss' (t5, t6 and t7) and predicate [B=1] and action 'B:=0' to the transitions 'timeout' (t3 and t4).

In the initial state M0 (only places p1, p3, and p7 each has a token), Sender is waiting for ACK0 (acknowledgement for the message with sequence number 0), while Receiver has sent ACK1 and is expecting for message 0. Suppose Receiver receives message 0 (by firing t10) and sends ACK0 (also by firing t10). The state of the PN changes to marking M1 (only places p1, p4 and p6 each has a token). On receiving ACK0,
Sender sends (by firing t2) message 1 and then waits for ACK1. Once Receiver gets (t11) message 1, it sends ACK1 (p4 gets a token). If Sender receives ACK1 (by firing t1), it transmits the next message 0. The system returns to the initial state of the next cycle.

When a message is lost, i.e., when Predicate [A=1] is true and one of transitions t5, t6 and t7 is fired, B is set to 1 (i.e., Action 'A:=0 and B:=1' is executed). Since transitions t3 and t4 ('time-out') are associated with Predicate [B=1], one of them will be fired, i.e., time is out, Sender retransmits the message.
5.1.2 Results of Analysis

The results of applying PNPDU to this protocol model are recorded below.

(1) Enumerative Analysis

Reachability Trees (shown in Figures 5-2, 5-3, 5-4 and 5-5):

Figure 5-2 shows the reachability tree for the case of a channel without message loss, i.e., A=B=0 during the execution of the PN. It shows that, starting from M0, it has only one path reaching the home-state, also M0. If a message is lost in the channel (i.e., variable A is set to 1), the reachability tree will be different from the one in Figure 5-2. It depends on the state (marking) in which the message is lost. Figures 5-3, 5-4 and 5-5 show the reachability trees for the cases where a message is lost in the first, second and third state, respectively. The reachability set of the PN is the combination of these trees.

Token-related properties (output of PNPDU):

THE RESULTS OF REACHABILITY ANALYSIS:
<1> THE NET IS SAFE.
<2> THE MAX NUMBER OF TOKENS IN EACH PLACE IS:
    (FOR COLOUR1) 1
<3> THE NET IS NOT CONSERVED
Figure 5-2: Reachability Tree for A Perfect Channel

Figure 5-3: The Reachability Tree When a MSG is Lost in the 1st state of Execution
Figure 5-4: The Reachability Tree When a MSG is Lost in the 2nd state of Execution

Execution-related Properties (output of PNPuo):

<4> EXECUTION-RELATED PROPERTIES:

NO DEADLOCKS.
NO LIVE-LOCKS.
NO LIVE-LOOPING.
M0 IS A HOME-STATE.

The logical correctness of the protocol can be verified by using the above output information. For example, the absence of deadlocks and livelocks in the trees shows that the protocol is free from these anomalies. Since M0 is both
Figure 5-5: The Reachability Tree When a MSG is Lost in the 3rd State of Execution

the initial state and a home-state, the protocol terminates correctly. There is no message overflow, as the PN (including the channel as a subset) is bounded.

(2) Structural Analysis

**Place Invariants** (output of PNPUO):

PLACE INVARIANT 1:
\[ M(1) + M(2) = 1 \]

PLACE INVARIANT 2:
\[ M(6) + M(7) = 1 \]

PLACE INVARIANT 3:
\[ M(1) + M(2) + M(6) + M(7) = 2 \]
These place invariants show, for example, that, during the execution of this PN, there is always a token in either p1 or p2, but not in both. The same situation is also true for p6 and p7. This means that in the alternating bit protocol, both entities (Sender and Receiver) have to stay in a single state, i.e., there is no state ambiguity of the protocol entity.

**Transition Invariants (output of PNPUSO):**

1. **Transition Invariant 1:**
   
   \[
   \begin{align*}
   & \text{T3} \quad \text{T5} \\
   & M \longrightarrow M
   \end{align*}
   \]

2. **Transition Invariant 2:**
   
   \[
   \begin{align*}
   & \text{T4} \quad \text{T7} \\
   & M \longrightarrow M
   \end{align*}
   \]

3. **Transition Invariant 3:**
   
   \[
   \begin{align*}
   & \text{T3} \quad \text{T4} \quad \text{T5} \quad \text{T7} \\
   & M \longrightarrow M
   \end{align*}
   \]

4. **Transition Invariant 4:**
   
   \[
   \begin{align*}
   & \text{T3} \quad \text{T6} \quad \text{T8} \\
   & M \longrightarrow M
   \end{align*}
   \]

5. **Transition Invariant 5:**
   
   \[
   \begin{align*}
   & \text{T3} \quad \text{T4} \quad \text{T6} \quad \text{T7} \quad \text{T8} \\
   & M \longrightarrow M
   \end{align*}
   \]

6. **Transition Invariant 6:**
   
   \[
   \begin{align*}
   & \text{T4} \quad \text{T6} \quad \text{T9} \\
   & M \longrightarrow M
   \end{align*}
   \]

7. **Transition Invariant 7:**
   
   \[
   \begin{align*}
   & \text{T3} \quad \text{T4} \quad \text{T5} \quad \text{T6} \quad \text{T9} \\
   & M \longrightarrow M
   \end{align*}
   \]
TRANSITION INVARIANT 8:
\[
\begin{align*}
& T1 \quad T2 \quad T8 \quad T9 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 9:
\[
\begin{align*}
& T2 \quad T3 \quad T5 \quad T8 \quad T9 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 10:
\[
\begin{align*}
& T1 \quad T2 \quad T4 \quad T7 \quad T8 \quad T9 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 11:
\[
\begin{align*}
& T1 \quad T2 \quad T3 \quad T4 \quad T5 \quad T7 \quad T8 \quad T9 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 12:
\[
\begin{align*}
& T1 \quad T2 \quad T10 \quad T11 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 13:
\[
\begin{align*}
& T1 \quad T2 \quad T3 \quad T5 \quad T10 \quad T11 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 14:
\[
\begin{align*}
& T1 \quad T2 \quad T4 \quad T7 \quad T10 \quad T'1 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 15:
\[
\begin{align*}
& T1 \quad T2 \quad T3 \quad T4 \quad T5 \quad T7 \quad T10 \quad T11 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 16:
\[
\begin{align*}
& T2 \quad T3 \quad T6 \quad T8 \quad T'0 \quad T11 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 17:
\[
\begin{align*}
& T1 \quad T2 \quad T3 \quad T4 \quad T6 \quad T7 \quad T8 \quad T10 \quad T11 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 18:
\[
\begin{align*}
& T1 \quad T2 \quad T4 \quad T6 \quad T9 \quad T10 \quad T11 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]

TRANSITION INVARIANT 19:
\[
\begin{align*}
& T1 \quad T2 \quad T3 \quad T4 \quad T5 \quad T6 \quad T9 \quad T10 \quad T'1 \\
& M \quad \text{\rightarrow} \quad M
\end{align*}
\]
These transition invariants indicate that there are 19 loops during the execution of this PN. They may be life-cycles (correct termination) or live loops, or even livelocks. The structural analysis process cannot distinguish them. It is necessary to check them in the reachability tree.

Transition invariant 1 implies that firing the transition sequence \((t_3, t_5)\) leads a marking (here it is the initial marking \(M_0\)) to itself again. This is obviously true for this PN. Correctness of the other invariants can also be checked in a similar way.

5.2 ANALYSING THE CONNECTION AND DISCONNECTION PHASES OF CLASS 1 AND CLASS 2 OF TRANSPORT PROTOCOL

This section describes our application of PNP to a portion of the Transport Protocol. We first use a basic PN model proposed by Berthelot [BBRT82] and compare his manually obtained results with ours. We then apply PNP again, using a predicate/action PN.

The Transport Protocol (TP), as proposed by both ISO (International Organization for Standardization) and ECMA (European Computer Manufacturers Association), has five different classes which handle errors of increasing severity [ECMA81] [ISO 87]. Each class provides services in three phases:
(1) Connection Establishment Phase
(2) Data Transfer Phase
(3) Disconnection Phase

The function of the Connection Establishment Phase is to provide the TP primitives: connect request, connect indication, connect response and connect confirmation for the establishment of an end-to-end transport connection between two transport entities. The Data Transfer Phase serves the exchange of data messages via TP primitives: data request, data indication, expedited-data request and expedited-data indication. The Disconnection Phase uses the TP primitives disconnect request, disconnect indication, disconnect response and disconnect confirmation to terminate a transport connection.

In the Class 1 protocol, not only are the functions of these three phases provided, the ability of error detection and reporting and of recovery from failures is also included. Class 2 provides also flow control for merging multiple Transport connections on a single Network connection.

Subsection 5.2.1 describes a basic PN model for the Connection and Disconnection Phases in Class 1 and Class 2 of the ECMA Transport Protocol. Subsection 5.2.2 gives the analysis results of applying PnPDU and compares them with those obtained manually by Berthelot [BERT82]. We also
apply PNPDUO to analyse a predicate/action PN model for this protocol in Subsection 5.2.3.

5.2.1 The BPN Model for Transport Protocol

In Berthelot's model (Figure 5-6), transition t₁ is shared by the two transport protocol entities and is used to indicate the reception of a connection request either from its Session Layer or from its peer Transport Protocol entity.

When t₁ is fired, each entity enters the Data Transfer State (p₂ and p₇ each get a token). Suppose, having transmitted data, one of the two entities sends (firing t₂ or t₆) a Disconnect Request (p₉ or p₄ gets a token). It then waits (p₃ or p₈) for Disconnection Confirm (waiting until p₅ or p₁₀ has got a token). If the other entity is still in a Data Transfer State (p₇ or p₂ has a token), it accepts (firing t₇ or t₂) the Disconnect Request and returns to its initial state, p₆ or p₁. Meanwhile, it sends Disconnection Confirm (p₅ or p₁₀ gets a token) to the requesting entity. The latter then returns (firing t₄ or t₈) to its initial state p₁ or p₆. Thus, a life-cycle is completed.

Another possible case occurs when one of the entities sends (firing t₂ or t₆) a Disconnect Request (p₉ or p₄ gets a token). The other entity, while waiting for (p₈ or p₃ has a token) a Disconnect Confirm, accepts (firing t₉ or t₅) the
Figure 5-6: A BPN Model for the Connection and Disconnection Phases of Transport Protocol

request and returns to the initial state (p6 or p1). The former receives (firing t5 or t9) the request (p4 or p9 has a token) and returns to the initial state p1 or p6.
5.2.2. Results of Analysis Based on a BPN Model

Following is the analysis results of applying PNPUS to the model discussed above.

(1) Enumerative Analysis

Reachability Tree (Figure 5-7):

```
M(1,0) (1 6)
  /       |
 t1       t2
  |
 M(2,0) (2 7)
       /       |
      t6       t7
      |
 M(3,0) (3 7 9)
       /       |
      t6       t2
      |
 M(3,1) (2 4 8)
       /       |
      t6       t2
      |
 M(3,2) (3 4 8 9)
       /       |
      t6       t2
      |
 M(4,1) (3 5 6)
       /       |
      t6       t2
      |
 M(4,2) (3 4 8 9)
       /       |
      t6       t2
      |
 M(4,3) (1 8 10)
```

Figure 5-7: Reachability Tree of the PN in Figure 5-6

Token-related properties (output of PNPUS):

THE RESULTS OF REACHABILITY ANALYSIS:
<1> THE NET IS SAFE.

<2> THE MAX NUMBER OF TOKENS IN EACH PLACE IS:

FOR COLOUR 1 1

<3> THE NET IS NOT CONSERVED

Execution-related Properties (output of PNPUS):

- 86 -
EXECUTION-RELATED PROPERTIES:
- No deadlocks
- No live-locks
- No live-looping
- No is a home-state

(2) Structural Analysis

Place Invariants (output of PNPUO):

PLACE INARIANT 1: \[ M(1) + M(2) + M(3) = 1 \]

PLACE INARIANT 2: \[ M(1) + M(4) + M(5) + M(7) = 1 \]

PLACE INARIANT 3: \[ M(6) + M(7) + M(8) = 1 \]

PLACE INARIANT 4: \[ M(1) + M(2) + M(3) + M(6) + M(7) + M(8) = 2 \]

PLACE INARIANT 5: \[ M(2) + M(6) + M(9) + M(10) = 1 \]

PLACE INARIANT 6: \[ M(1) + M(2) + M(4) + M(5) + M(6) + M(7) + M(9) + M(10) = 2 \]

Transition Invariants (output of PNPUO):

TRANSITION INARIANT 1: 

\[
\begin{array}{cccc}
T1 & T2 & T4 & T7 \\
M & \rightarrow & M \\
\end{array}
\]
TRANSITION INVARIANT 2:

\[
\begin{array}{c}
T_1 \ T_3 \ T_6 \ T_8 \\
M \longrightarrow M
\end{array}
\]

TRANSITION INVARIANT 3:

\[
\begin{array}{c}
T_1 \ T_2 \ T_5 \ T_6 \ T_9 \\
M \longrightarrow M
\end{array}
\]

From these results, it can be concluded that the protocol is free from deadlocks and livelocks and that it terminates correctly at the initial state. As the PN is safe and bounded, there is no channel overflow.

The place invariants (1), (3), (7), (5) we obtain are the same as those of [BERT82]. The other two, (4) and (6), are also correct, but do not appear in [BERT82]. Here, (1) and (3) show that each entity stays in one of the three phases at any moment. (2) and (5) indicate that, if one entity is in the Disconnection Phase (p_1 or p_6), the other cannot be in the Data Transfer Phase (p_7 or p_2) and cannot issue a Disconnect Request (p_4 or p_9) or Disconnect Confirm (p_{10} or p_5) primitive. Place invariant (4) is implied by (1) and (3), and place invariant (6) by (2) and (5).

All our transition invariants are the same as those obtained by Berthelot [BERT82] manually. The first one indicates that firing the sequence of transitions t_1, t_3, t_4, t_7 leads a marking M (here M_0) to itself (M_0). The same situations occur for the sequence t_1, t_3, t_5, t_8 and sequence t_1,t_2, t_5, t_6, t_9. These invariants are reflected
in the reachability tree (Figure 5-7) as those paths starting from the root \( M(0) \) and terminating at the leaves which are home-states. Note that occasionally different paths in the tree may correspond to the same sequence of firing transitions. For example, the third transition invariant corresponds to the two leftmost paths in the tree:

(i) \( M(1,0) \rightarrow M(2,0) \rightarrow M(3,0) \rightarrow M(4,0) \rightarrow M(5,0) \rightarrow M(6,0) \)

(ii) \( M(1,0) \rightarrow M(2,0) \rightarrow M(3,0) \rightarrow M(4,0) \rightarrow M(5,1) \rightarrow M(6,1) \).

5.2.3 Results of Analysis Based on A Predicate/action PN

For more efficient analysis, we have also applied PNPDUO to verify the Transport Protocol using a predicate/action PN. We associate a predicate, say \( [A=1] \), with transition \( t_1 \), to indicate that the demanded quality of the connection is met. The initial value of \( A \) is 0. Two predicates, \( [B=1] \) and \( [C=1] \), are attached to \( t_2 \) and \( t_6 \), to represent the end of the Data Transfer Phase in each entity. The corresponding actions, \( A:=0 \), \( B:=0 \) and \( C:=0 \), are related to transitions \( t_1 \), \( t_2 \) and \( t_6 \), respectively. This model is shown in Figure 5-8.

We got the following analysis results for this model. If we assign the value 1 to the variables \( A \), \( B \), and \( C \) at the beginning of the reachability analysis, we get the same reachability tree as Figure 5-7. This means that if the protocol entities requires connection and data transfer, the protocol works and terminates properly.
Figure 5-8: A Predicate/Action PN Model for the Transport Protocol

If we do not give new values to these variables (the system keeps A=B=C=0) during the generation of the reachability tree, we get the outputs as shown in Figure 5-9. This result indicates that if the Session layer of the peer entity does not issue any Connection Request, the Transport Protocol will not enter the Connection Phase.

Figure 5-10 shows the reachability tree in the case of assigning A=1 at the beginning of executing the PN and keeping B=C=0 during the execution. This is the situation when both entities stay in the Data Transfer Phase for ever.
If we set $A=1$ and $B=1$ but keep $C=0$ during the execution, we get the reachability tree as shown in Figure 5-11. This figure shows that one of the entities is forced back to the Disconnection Phase before it has finished transferring its data.

For structural analysis, we get the same results as for the BPN model (Subsection 5.2.2).

\[
M(1,0) \begin{array}{c} 1 \\ 6 \end{array}
\]

\[
A=B=C=0
\]

\[---\text{deadlock}\]

**Figure 5-9:** The Reachability Tree for the improved Model with $A=B=C=0$

\[
M(1,0) \begin{array}{c} 1 \\ 6 \end{array}
\]

\[
A=1 \quad B=C=0
\]

\[
t1
\]

\[
M(2,0) \begin{array}{c} 2 \\ 7 \end{array}
\]

\[---\text{deadlock}\]

**Figure 5-10:** The Reachability Tree for the Improved Model with $A=1$ and $B=C=0$
Figure 5-11: The Reachability Tree for the Improved Model with A=B=1 and C=0
Chapter VI

CONCLUSION AND SUGGESTIONS FOR FURTHER RESEARCH
AND DEVELOPMENT

In this thesis, we have introduced the basic concepts and properties of Petri nets. A brief review of the application of Petri net theory on the modeling and verification of communication protocols were summarized. Independent of OGIVE, we have developed a software package, PNPVO, for specifying and analyzing logical properties of systems modeled in terms of Petri nets. It is implemented in PASCAL 8000 under CMS operating system on the AMDAHL computer. Its program requires about 10 kwords of memory space. For a Petri net with 40 places and 40 transitions, it is necessary to have another 10 kwords space to contain the data. Its objectives, user’s interface, and implementational details were presented. Usefulness of the package for protocol verification was shown in the analysis results of two well-known protocols: the alternating bit protocol and a transport protocol.

In an efficient way, PNPVO provides a lot of convenience to the user as an automated aid for analyzing many properties, such as reachability trees, deadlocks, livelocks, home-states, safeness, boundedness and
invariants. The user does not have to know the transition firing rules, the techniques for generating reachability trees, or the algebra of solving incidence equations.

Though the current version of PNPUO is for token-coloured and coloured predicate/action PN only, other types of PN can be handled quite easily. For timed PN, the time interval \([t^*, t^{**}]\) can be represented by a predicate like \([t > t^* \text{ and } t < t^{**}]\). This kind of predicates and the corresponding actions like \(t := t + t^{**}\) should be related to each transition of the net, where \(E\) is a time variable whose initial value is 0. In order to perform 'timeout' function in a timed PN, we can give larger value of \(t^{**}\) with transition 'timeout' than that with other transitions, but less than the sum of any two of them. In this case, the format of predicates in PNPUO has to extend to including '>', 'large' and '<', 'less' expressions.

Petri nets with inhibitor arcs can be converted into basic PN by adding pairs of places and transitions as described in [MOA78]. The resulting PN can be analysed by PNPUO.

To enhance PNPUO, it can be extended in several aspects:

(1) The maximum numbers of places, transitions, colours, and variables of the Petri nets allowed in the current version of PNPUO can be changed arbitrarily by setting the corresponding program parameters \(m, n, l\) etc.)
(2) Other types of PN can be handled, possibly without conversion first, by adding into the current version of PNPWO procedures similar to the existing ones for handling input data and recognizing predicates and actions.

(3) Graphics capability may be added to PNPWO. The current PNPWO has no facilities for drawing the input Petri net diagrams or the subsequent reachability trees on the screen or on to hard copies. Graphics features will greatly enhance its visible capability.
REFERENCES


COOL83 J. E. Coolahan, JR. and N. Roussopoulos, "Timing Requirements for Time-Driven Systems Using


MOAL78  M. Moalla, J. Pulou and J. Sifakis, 'Synchronized Petri Nets: a Model for the Description of Non-


Appendix A
PROGRAM LISTINGS OF PNPUD
(* NAME : PACKAGE *)
(* FUNCTION : This is a software package for storing, *)
(*       displaying, analysing coloured pridets *)
(*       action Petri nets. *)
(* INPUT FILES : DATAIN DATA -- numerical data *)
(*       DIN DATA -- character data *)
(* OUTPUT FILES : OUTIN DATA -- numerical data *)
(*       OUT DATA -- character data *)
(*       DATAOUT DATA -- the record of the *)
(*       complete interactive process *)
(* PROCEDURES CALLED : NINTO *)
(*       CWRITE *)
(*       CREATE *)
(*       DISPLAY *)
(*       ANALYSIS *)

******************************************************************************
PROGRAM PACKAGE(INPUT/, OUTPUT, DATAIN,DATOUT,OUTIN,DIN,OUT) ;

TYPE LIST = PACKED ARRAY (.1..40 , .1..9.) OF INTEGER;
CB  = PACKED ARRAY (.1..40 , .1..40 , .1..9.) OF INTEGER;
CC  = PACKED ARRAY (.1..8 , .1..81.) OF CHAR ;
CD  = PACKED ARRAY (.1..40, .1..81.) OF CHAR ;

VAR PARC, TARC : CB ;
   P : LIST ;
   COM, COM1, COM2 : CC ;
   PP, PT, AP, AT : CD ;
   N,M,L,N1,N2,N3 : INTEGER ;
   A B,C,D,E F,G,H : INTEGER ;
   A*,B1,C1,D1,E1,F1,G1,H1 : REAL ;
   A?,B2,C2,D2,E2,F2,G2,H2 : BOOLEAN ;
   T : CHAR ;
   R FLAG,DFLAG,MFLAG : INTEGER ;
   DATAIN, DATOUT, OUTIN : TEXT ;
   DIN OUT : TEXT ;

PROCEDURE ANALYSIS (VAR P:LIST; VAR PARC,TARC: CB;)
   VAR PP,PT AP,AT:CD;VAR M,N,L:INTEGER ) ;

EXTERN ;
   (* procedure analysis is external *)

******************************************************************************
(* NAME : CWRITE *)
(* FUNCTION : Write numerical and character data *)
(*       from variables and arrays into two *)
(*       output files "OUTIN DATA" and "OUT *)
(*       "DATA", respectively, for subse- *)
(*       quent use. *)
(* DATA RESOURCE : program variables and arrays *)

PROCEDURE CWRITE ;
   VAR I J,K : INTEGER ;

BEGIN
   REWRITE 'OUTIN) ;
   (* Open file "OUTIN DATA" *)
   REWRITE (OUT) ;
   (* Open file "OUT DATA" *)
   WRITELN (OUTIN,M:2,' ',N:2,' ',L:1) ;
   (*
   (* write arcs P->T -- PARC -- MxNxL matrix *)
   - 102 -

******************************************************************************
(* into OUTIN DATA *)

(* FOR I:=1 TO M DO *)
FOR J:=1 TO N DO
BEGIN
FOR K:=1 TO L DO
   WRITE (OUTIN,PARC(.I,J,K.) :2, ' ') ;
   WRITELN (OUTIN)
END ;

(* Write arcs T->p -- TARC -- MxNxL matrix *)
(* into OUTIN DATA *)

(* FOR J:=1 TO N DO *)
FOR I:=1 TO M DO
BEGIN
FOR K:=1 TO L DO
   WRITE (OUTIN,TARC(.I,J,K.) :2, ' ') ;
   WRITELN (OUTIN)
END ;

(* Write names of user's integer variables -- COM *)
(* -- character strings into OUT DATA *)

(* FOR I:=1 TO 8 DO *)
BEGIN
   J := 1 ;
   WHILE COM(.I,J.)<>"$" DO
   BEGIN
      WRITE (OUT,COM(.I,J.) ) ;
      J := J+1
   END ;
   WRITELN (OUT,"$")
END ;

(* Write names of user's real variables -- COM1 *)
(* -- character strings into OUT DATA *)

(* FOR I:=1 TO 8 DO *)
BEGIN
   J := 1 ;
   WHILE COM1(.I,J ) <> "$" DO
   BEGIN
      WRITE (OUT,COM1(.I J ) ) ;
      J := J+1
   END ;
   WRITELN (OUT,"$")
END ;

(* Write names of user's Boolean variables -- COM2 *)
(* -- character strings into OUT DATA *)

(* FOR I:=1 TO 8 DO *)
BEGIN
J := 1;
WHILE COM2(I,J) <> "" DO
BEGIN
  WRITE (OUT,COM2(I,J)) ;
  J := J+1;
END;
PRINTLN (OUT,"$")
END;

(*
(* Write the predicates on transitions -- PT *
(* -- character strings into OUT DATA *
(*
FOR I:=1 TO N DO
BEGIN
  J := 1;
  WHILE PT(I,J) <> "" DO
  BEGIN
    WRITE (OUT,PT(I,J)) ;
    J := J+1
  END;
  PRINTLN (OUT,"$")
END;

(*
(* Write the actions at transitions -- AT *
(* -- character strings into OUT DATA *
(*
FOR I:=1 TO N DO
BEGIN
  J := 1;
  WHILE AT(I,J) <> "" DO
  BEGIN
    WRITE (OUT,AT(I,J)) ;
    J := J+1
  END;
  PRINTLN (OUT,"$")
END;

(*
(* Write the predicates on places -- PP *
(* -- character strings into OUT DATA *
(*
FOR I:=1 TO M DO
BEGIN
  J := 1;
  WHILE PP(I,J) <> "" DO
  BEGIN
    WRITE (OUT,PP(I,J)) ;
    J := J+1
  END;
  PRINTLN (OUT,"$")
END;

(*
(* Write the actions on places -- AP *
(* -- character strings into OUT DATA *
(*
- 104 -
FOR I:=1 TO M DO
BEGIN
    J := 1;
    WHILE AP(.I,J.) <> "$" DO
    BEGIN
        WRITE (OUT,AP(.I,J.));
        J := J+1
    END;
    WRITELN (OUT, "$")
END;
(*
(* Write initial marking M0 -- P *
(* -- mxml matrix into OUTIN DATA *)
(*
(* FOR I:=1 TO M DO
BEGIN
    FOR J:=1 TO L DO
    WRITE (OUTIN,P(.I,J.):2,,"\"));
    WRITELN (OUTIN)
END;
(*
(* Write the numbers of user's integer, real, and *
(* Boolean variables into OUTIN DATA *)
(*
(* WRITELN (OUTIN,"\",N1:1,"\",N2:1,"\",N3:1));
(*
(* Write the value of each integer variable *
(* into OUTIN DATA, if there are some *)
(*
(* IF N' <> 0 THEN
    FOR I:=1 TO N1 DO
    CASE I-1 OF
        0 : WRITELN (OUTIN A:2) ; 3 : WRITELN (OUTIN,B:2) ;
        1 : WRITELN (OUTIN,C:2) ; 4 : WRITELN (OUTIN,E:2) ;
        5 : WRITELN (OUTIN,D:2) ; 6 : WRITELN (OUTIN,F:2) ;
        END;
    FLAG := 1 (* Remember PN data has already stored in files*)
END;
(*-------------------------------------------------------)
(* NAME : CREATE *)
(* FUNCTION : To accept user's data of a PN and store them into the following Vars *)
(* and arrays *)
(* PROCEDURES CALLED : INTRO *)
(* CDECLARATION *)
(* CPREDACT *)
(* CINITIAL *)
(* CWRITE *)
(* PROGRAM VARIABLES : *)
(* M -- the total number of places in the PN <integer> *)
(* N -- the total number of transition in PN <integer> *)
(* L -- the total number of token colours <integer> *)
(* PROGRAM ARRAYS : *)
PROCEDURE CREATE ;
VAR I, J, K, R : INTEGER ;
T, CH : CHAR ;

PROCEDURE CDECLARATION ;
(* NAME : ) CNUMBER
(* FUNCTION : ) Accept the total numbers of places, transitions, and colours.
(* ) store them in Vars M, N, and L .

PROCEDURE CNUMBER ;
VAR I : INTEGER ;
BEGIN
WRITE ('* ENTER TOTAL NO. M OF PLACES *');
WRITELN ('< NO BIGGER THAN 40 OF YOUR NET *');
READLN ;
READ(M) ;
WRITELN (DATAOUT,' TOTAL NO. M OF PLACES M="',M:2);
WRITE ('* ENTER TOTAL NO. N OF TRANSITIONS *');
WRITELN ('< NO BIGGER THAN 40 OF YOUR NET *');
READLN ;
READ(N') ;
WRITELN (DATAOUT,' TOTAL NO. N OF TRANSITIONS N="',N:2);
WRITELN (DATAOUT, ** TOTAL NO OF TOKEN TYPES \L=^,L:2 \)**
WRITELN ;
WRITELN (DATAOUT) ;

(* If did not store any data of this PN into the two *)
(* output files "OUTIN" and "OUT", *)
(* then initialize the numbers of each type of variables*)
(* <n1,n2 n3 for integer, real, Boolean resp. > *)
(* and all the arrays. *)
(*

IF FLAG=0 THEN
BEGIN
N*:0 ; N2*:0 ; N3*:0 ;
FOR I:=1 TO 8 DO
BEGIN
    COM (.I,1.) := "\$" ;
    COM1 (.I,1.) := "\$" ;
    COM2 (.I,1.) := "\$"
END ;
FOR I:=1 TO N DO
BEGIN
    PT (.I,1.) := "\$" ;
    AT (.I,1.) := "\$
END ;
FOR I:=1 TO M DO
BEGIN
    PP (.I,1.) := "\$" ;
    AP (.I 1.) := "\$
END
END

*****************************************************************************
(* NAME : CIRC *)
(* FUNCTION : Input all the numbers of arcs from *)
(* places to transitions P->T, and *)
(* from transitions to places T->P . *)
*****************************************************************************
PROCEDURE CIRC ;
VAR I J,K,K1 : INTEGER ;

BEGIN
    (** Write the title in each output file **)
    (** and the terminal **) (**
    WRITELN (DATAOUT, ** ARCS P ---> T & T ---> P **));
    WRITELN (DATAOUT) ;
    WRITELN (** SPECIFY TOKEN NUMBER FOR EACH COLOUR TYPE **);
    WRITE (** PLACE TO TRANS. **);
    WRITE (DATAOUT, ** PLACE TO TRANS. **);
    FOR K:=1 TO L DO
    BEGIN
        WRITE (** COL,K:1 **);
        WRITE (DATAOUT, ** COL,K:1)
    END ;
WRITELN;
WRITELN (DATAOUT);
WRITE("--------------" );
WRITE (DATAOUT,"--------------" );
FOR K:=1 TO L DO
  BEGIN
    WRITE("-------");
    WRITE (DATAOUT,"-------");
  END;
WRITELN;
WRITELN;
WRITELN (DATAOUT);
(*
(* input arcs P->T
(*
(*
FOR I:=1 TO M DO
FOR J:=1 TO N DO
BEGIN
  IF I<= 9 THEN BEGIN (* write Pi *)
    WRITE(" P0",I:1);
    WRITE (DATAOUT," P0",I:1)
  END
  ELSE BEGIN
    WRITE(" P",I:2);
    WRITE (DATAOUT," P",I:2)
  END;
  IF J<=9 THEN BEGIN (* write Tj *)
    WRITELN(" T0",J:1);
    WRITE (DATAOUT," T0",J:1)
  END
  ELSE BEGIN
    WRITELN(" T",J:2);
    WRITE (DATAOUT," T",J:2)
  END;
READLN;
(*
(* input Pi-> Tj
(*
(*
FOR K:=1 TO L DO
BEGIN
  READ(R); (* input Pi->Tj in colour k *)
  PARC(.I,J,K.) := R
END;
WRITE(" ");
(*
(* display Pi->Tj
(*
(*
FOR K:=1 TO L DO
BEGIN (* on terminal and in file *)
  WRITE(" ",PARC (.I,J,K.):2);
  WRITE (DATAOUT," ",PARC (.I,J,K.):2)
END;
WRITELN;
WRITELN;
WRITELN (DATAOUT);  
WRITELN (DATAOUT)  
END;  
(* write title in each file and on the terminal *)  
(*                                     *)  
WRITELN;  
WRITELN;  
WRITELN (DATAOUT);  
WRITELN (DATAOUT);  
WRITE ("TRANS. TO PLACE");  
WRITE (DATAOUT, "TRANS. TO PLACE");  
FOR K:=1 TO L DO  
BEGIN  
  WRITE (" COL", K:1);  
  WRITE (DATAOUT, " COL", K:1)  
END;  
WRITELN (DATAOUT);  
WRITELN;  
WRITE (" ---------------------");  
WRITE (DATAOUT, " ---------------------");  
FOR K:=1 TO L DO  
BEGIN  
  WRITE (" ----");  
  WRITE (DATAOUT, " ----");  
END;  
WRITELN (DATAOUT);  
WRITELN (DATAOUT);  
WRITELN;  
WRITELN;  
(* input arcs T->P *)  
(*                   *)  
(*                   *)  
FOR J:=1 TO N DO  
FOR I:=1 TO M DO  
BEGIN  
  IF J<=9 THEN BEGIN (* write Tj *)  
    WRITE (" T0", J:1);  
    WRITE (DATAOUT, " T0", J:1)  
  END  
  ELSE BEGIN  
    WRITE (" T", J:2);  
    WRITE (DATAOUT, " T", J:2)  
  END;  
  IF I<=9 THEN BEGIN (* write Pi *)  
    WRITELN (" P0", I:1);  
    WRITE (DATAOUT, " P0", I:1)  
  END  
  ELSE BEGIN  
    WRITELN (" P", I:2);  
    WRITE (DATAOUT, " P", I:2)  
  END;  
READLN;  
(*                   *)  
(*                   *)
BEGIN
READ(R);  (* input Tj->Pi in colour k*)
TARC(.I,J,K.) := R
END;
WRITE (''
(*    display    Tj -> Pi
(*
(*
FOR K:=1 TO L DO
BEGIN
(* on terminal and in file   *)
WRITE('', TARC(.I,J,K.):2);
WRITE(DATAOUT, ''
TARC(.I,J,K.):2)
END;
WRITELN(DATAOUT);
WRITELN(DATAOUT);
WRITELN;
WRITELN
END
END;
******************************************************************************************
(* NAME : CINTEGER
(* FUNCTION : Accept names of user's integer variables, *
(*         store them in array 'COM'.
---------******************************************************************************************
PROCEDURE CINTEGER;
VAR I J : INTEGER;
BEGIN
WRITELN ('* * DEFINE PROGRAM INTEGER VARIABLES * *').;
WRITELN (DATAOUT, '* * INTEGER VARIABLES * *');
WRITELN ('* ENTER THE TOTAL NUMBER OF INTEGER *');
WRITELN ('* VARIABLES < NOT BIGGER THAN 8 > :');
READLN;
READ(N1);(*input the total number of integer variables*)
WRITELN('DATAOUT);
(*
(*     input all the names of user's integer variables *
(*
(*
FOR I:=1 TO N' DO
BEGIN
CH :=CHR(I+ORD('A')-1);
WRITE ('* ENTER YOUR NO. ',I:1);
WRITELN ('* INTEGER VARIABLE NAME: ');
READLN;
(*
(*     input the ith variable name *
(*
(*
J := 1;
WHILE NOT EOLN DO
BEGIN
READ(T);  (* input jth char. in ith name *)
COM (.I,J.) := T;
- 110 -
J := J + 1
END;
(* add the suffix "$" in ith name *)
COM(.I,J.) := "$";
(*                    *)
(* display ith name of integer name *)
(*                    *)
J := 1;
WRITE ("    ");
WRITE (DATAOUT,"   ,CH:2." : "");
WHILE COM(.I,J.) <> "$" DO
BEGIN
  WRITE (COM(.I,J.));          (* on terminal *)
  WRITE (DATAOUT,COM(.I,J.)); (* on file    *)
  J := J + 1
END;
WRITELN;
WRITELN (DATAOUT)
END

*******************************************************************************
(* NAME : CREAL                 *)
(* FUNCTION : Accept all the names of user's real variables and store them into array "COM1". *)
*******************************************************************************

PROCEDURE CREAL;
VAR I J : INTEGER;
T : CHAR;
BEGIN
  WRITELN (" ** DEFINE PROGRAM REAL VARIABLES " ");
  WRITELN;
  WRITELN (DATAOUT);          
  WRITELN (DATAOUT, " ** REAL VARIABLES " ");
  WRITELN (DATAOUT);
  WRITELN (" * ENTER THE TOTAL NUMBER OF REAL VARIABLES < NO BIGGER THAN 8 > : ");
  READLN;
  READ (N2);          (* input the total num. of real variables *)
  FOR I:=1 TO N2 DO
    BEGIN
      CH := CHR ( I+ORD('A')-1 );
      WRITELN (" * ENTER NO. ,I:1, REAL VARIABLE NAME:" );
      READLN;
      (*                     *)
      (* input ith real variable name  *)
      (*                     *)
      J := 1;
      WHILE NOT EOLN DO
        BEGIN
          READ (T);         (*input jth character in ith name*)
          COM1(.I J) := T ;
          J := J + 1
        END;
      WRITELN ("    ");
COM1(.I,J) := "$" ; (* add suffix in ith name *)
(*
* display ith real variable name
*)
(*
J := 1;
WRITE ("");
WRITE (DATAOUT,"CH:2,1:1");
WHILE COM1(.I,J) <> "$" DO
BEGIN
WRITE (COM1(.I,J)); (* on terminal *)
WRITE (DATAOUT,COM1(.I,J)); (* on file *)
J := J+1;
END;
WRITELN;
WRITELN(DATAOUT);
END
END;

******************************************************************************
(* NAME : CLOGIC *)
(* FUNCTION : Accept all names of user's Boolean *)
(* variables store them into array 'COM2'. *)
******************************************************************************
PROCEDURE CLOGIC;
VAR I J : INTEGER;
T : CHAR;
BEGIN
WRITELN ("*** DEFINE PROGRAM BOOLEAN VARIABLES ***");
WRITELN;
WRITELN (DATAOUT);
WRITELN (DATAOUT,"*** BOOLEAN VARIABLES ***");
WRITELN (DATAOUT);
WRITELN ("ENTER THE TOTAL NUMBER OF BOOLEAN ");
WRITELN ("VARIABLES < NO BIGGER THAN 8 > : ");
READLN;
READ (N3); (*input the total num. of Boolean variables*)
FOR I:=1 TO N3 DO
BEGIN
CH := CHR(I+ORD("A")-1);
WRITE ("ENTER YOUR NO.,I:1," BOOLEAN VARIABLE");
WRITELN ("NAME : ");
READLN;
(*
* input ith Boolean variable name
*)
(*
J := 1;
WHILE NOT EOLN DO
BEGIN
READ (T); (* input jth character in ith name *)
COM2(.I,J.) := T;
J := J+1
END
WRITE (" ");
(* add suffix "$" for each name *)
COM2(.I,J ) := "$";
(* display ith Boolean variable name *)

J := 1;
WRITE (''),
WRITE (DATAOUT,'',CH:='2 : '),
WHILE COM2(I,J)<>'$', DO
BEGIN
WRITE (COM2(I,J)); (* on terminal *)
WRITE (DATAOUT,COM2(I,J)); (* on file *)
J := J+1
END;
WRITELN;
WRITELN(DATAOUT)
END;

END;

********************************************************************************
(* NAME : FIRST *)
(* FUNCTION : contain the options of entity declaration *)
********************************************************************************

PROCEDURE FIRST;
BEGIN
WRITELN('** ENTITY DECLARATION PHASE **');
WRITELN('< ON CREATE >');
WRITELN(' -------------------------------');
WRITELN(' * DO YOU WANT TO DECLARE : ');
WRITELN(' * 1. NUMBERS OF PLACES, TRANSITIONS, *');
WRITELN(' * AND COLOUR TYPES *');
WRITELN(' * 2. ARCS P--->T & T--->P *');
WRITELN(' * 3. INTEGER VARIABLES *');
WRITELN(' * 4. REAL VARIABLES *');
WRITELN(' * 5. BOOLEAN VARIABLES *');
WRITELN(' -------------------------------');
WRITELN(' * ENTER YOUR SELECTION *');
WRITELN(' * HIT ANY DIGITS OTHER THEN 1, 2, 3, 4, 5 *');
WRITELN(' GET OUT OF ENTITY DECLARATION PHASE *');
END;

********************************************************************************
(* NAME : INTO *)
(* FUNCTION : display the options of entity declaration *)
(* on the terminal *)
(* PROCEDURES CALLED : FIRST *)
********************************************************************************

PROCEDURE INTO;
BEGIN
T:="N";
WHILE T <> "Y" DO
BEGIN
FIRST; (* invoke procedure FIRST *)
READLN;
READ (R); (* input user's selection *)
WRITELN(' <', R:1, '> OK ? --- Y OR N');
READLN;
READ (T) (* input user's Y/N decision *)
END;
END
END;
(*
(* The main program of procedure CDECLARATION *)
(*
BEGIN
WRITELN(DATAOUT, " ** ENTITY DECLARATION **");
WRITELN (DATAOUT) ;
INTO ; (* invoke procedure INTO *)
WHILE (R>0) AND (R<6) DO
BEGIN
IF R=1
THEN CNUMBER (* call procedure CNUMBER *)
ELSE IF R=2
THEN CARC (* call procedure CARC *)
ELSE IF R=3
THEN CINTEGER (* call CINTEGER *)
ELSE IF R=4
THEN CREAL (* call *)
ELSE CLOGIC ; (* call *)
INTO (* call procedure INTO .*)
END
END;

************************************************************************************
(* NAME : CPREDACT *)
(* FUNCTION : Accept all the predicates and actions and *)
(* store them into arrays PT AT, PP,AP. *)
(* PROCEDURE CALLED : RECOM *)
(* CPREDTR *)
(* CACTTR *)
(* CPREDPL *)
(* CACTPL *)
*************************************************************************************
PROCEDURE CPREDACT ;
VAR R I : INTEGER ;

************************************************************************************
(* NAME : CACTPL *)
(* FUNCTION : Accept all actions at all the places and *)
(* store them into array 'AP'. *)
*************************************************************************************
PROCEDURE CACTPL ;
VAR I,J : INTEGER ;
T : CHAR ;
BEGIN
WRITELN (" * * ENTER ACTION AT EACH PLACE **");
WRITELN (" * IN THE FORMAT AS A:=A+1 **");
WRITELN (DATAOUT);
WRITELN (DATAOUT," ** ACTIONS AT PLACES **");
WRITELN (DATAOUT);
WRITELN ;
FOR I:=1 TO M DO
BEGIN
(* write "Pi :" on terminal and record file *)
IF I<=9 THEN BEGIN
(A).
- 114 -
WRITE\( \left( \text{"} P^I: 1 \text{"}, \text{"} : \text{"} \right) \); 
WRITE\( \left( \text{"} \text{P}^I: 1 \text{"}, \text{"} : \text{"} \right) \); 
END
ELSE BEGIN
WRITE\( \left( \text{"} P^I: 2 \text{"}, \text{"} : \text{"} \right) \);
WRITE\( \left( \text{"} \text{P}^I: 2 \text{"}, \text{"} : \text{"} \right) \);
END
READLN;
(*
(* input the action at \( \text{ith} \) place *)
(*
*)
J := 1;
WHILE NOT EOLN DO
BEGIN
\(*\) input jth char. of action at ith place \(*\)
READ\( \left( \text{T} \right) \);
AP\( (.I, J ) \) := T;
J := J + 1
END
\(*\) add suffix \"\( \$ \)\" in the action at ith palce \(*\)
AP\( (.I, J ) \) := \"\( \$ \)\";
\(*\) \(*\)
\(*\) display the action at ith place \(*\)
(*
*)
J := 1;
WRITE\( \left( \text{"} \right) \);
WRITE\( \left( \text{DATAOUT, \"} \right) \);
WHILE AP\( (.I, J ) \) <> \"\( \$ \)" DO
BEGIN
WRITE\( \left( \text{DATAOUT,AP\( (.I, J ) \) \}} \right) \); (* write on file *)
J := J + 1
END
WRITE\( \left( \text{DATAOUT} \right) \);
END
END

(NAME : CACTTR) (*
(FUNCTION : Accept all the action at all the tran- (*)
(sitions and store them into array \"AT\" . \(*

(CACTTR)
PROCEDURE CACTTR ;
VAR I, J : INTEGER ;
T : CHAR ;
BEGIN
WRITE\( \left( \text{"} \text{** ENTER ACTION AT EACH TRANSITION} \text{"} \right) \);
WRITE\( \left( \text{"} \text{IN THE FORMAT AS A := A+1} \text{"} \right) \);
WRITE\( \left( \text{DATAOUT} \right) \);
WRITE\( \left( \text{DATAOUT} \right) \);
WRITE\( \left( \text{DATAOUT} \right) \);
WRITE\( \left( \text{DATAOUT} \right) \);
WRITE\( \left( \text{DATAOUT} \right) \);
FOR I := 1 TO N DO
BEGIN
(* write 'Ti : ' on terminal and record file *)
IF I<9 THEN BEGIN
    WRITELN(``, T0`,I:=1, = `);
    WRITE (DATAOUT, ``, T0`,I:=1, = `)
END
ELSE BEGIN
    WRITELN(``, T`,I:=2, = `);
    WRITE (DATAOUT, ``, T`,I:=2, = `)
END

READLN;
(*
(* input the action at ith transition *)
(*
J := 1;
WHILE NOT EOLN DO BEGIN
    (* input jth char. of the action at ith trans. *)
    READ (T);
    AT(I,J) := T;
    J := J+1
END;
(* ass suffix `$` in the action at ith transition *)
AT(I,J) := `$`;
(* display the action at ith transition *)
J := 1;
WRITE (`
WRITE (DATAOUT, ``
WHILE AT(I,J) <> `$` DO BEGIN
    WRITE(AT(I,J));  (* display on terminal *)
    WRITE (DATAOUT,AT(I,J));  (* write into record file *)
    J := J+1
END;
WRITELN;
WRITELN (DATAOUT)
END END;

(*-----------------------------------------------------------------------*)
(* NAME : CPREDPL *)
(* FUNCTION : Accept all the predicates on all the *)
(* places and store them into array 'PP'. *)
(*-----------------------------------------------------------------------*)

PROCEDURE CPREDPL ;
VAR I J : INTEGER ;
T : CHAR ;
BEGIN
WRITELN(`` ** ENTER PREDICATES ON EACH PLACE ** '``);
WRITELN(`` ** IN THE FORMAT AS A=0 ** '``);
WRITELN (DATAOUT);
WRITELN (DATAOUT, ` ` ** PREDICATES ON PLACES ** '``);
WRITELN (DATAOUT);
WRITELN;
FOR I:=1 TO M DO BEGIN
(* write 'Pi : ' on terminal and record file *)
IF I<=9 THEN BEGIN
  WRITELN (' P0',I:1, ' : ') ;
  WRITE (DATAOUT, ' P0',I:1, ' : ')
END
ELSE BEGIN
  WRITELN (' P',I:2, ' : ');
  WRITE (DATAOUT, ' P',I:2, ' : ')
END;

READLN ;
(*
(* input the predicate on place Pi *)
(*
J := 1;
WHILE NOT EOLN DO
BEGIN
  (* input jth char. of the predicate on ith place*)
  READ (T);
  PP(.I J ) := T;
  J := J+1
END;
PP(.I J ) := '$' ; (* add suffix to the ith predicate*)
(* display the predicate on ith place *)
J := 1;
WRITE (' ',
WRITE (DATAOUT,' ');
WHILE PP(.I,J )<>'$' DO
BEGIN
  WRITE (PP(.I,J ));
  WRITE (DATAOUT,PP(.I,J ));
  (* display on terminal *)
  J := J+1
END;
WRITELN;
WRITELN (DATAOUT)
END;

*******************************************************************************
(* NAME : CPREDTR *)
(* FUNCTION : To accept all the predicates on all the transitions and store them into array 'PT' *)
*******************************************************************************
PROCEDURE CPREDTR ;
VAR I,J : INTEGER ;
T : CHAR ;
BEGIN
WRITELN (' * ENTER PREDICATES ON EACH TRANSITION * ');
WRITELN (' * IN THE FORMAT AS A=0 * ');
WRITELN (DATAOUT);
WRITELN (DATAOUT, ' * * PREDICATES ON TRANSITIONS * * ');
WRITELN (DATAOUT);
WRITELN;
FOR I:=1 TO N DO
BEGIN
  (* write 'Ti : ' on terminal and record file *)
  WRITE (DATAOUT,' Ti : ');
  Write (DATAOUT, ' Ti : ')
END;
IF I<=9 THEN BEGIN
  WRITELN (" TO", I:1, " : ");
  WRITE (DATAOUT," TO", I:1, " : ");
END
ELSE BEGIN
  WRITELN (" T", I:2, " : ");
  WRITE (DATAOUT," T", I:2, " : ");
END;

READLN;
(* input the predicate on ith transition *)
(*
J := 1;
WHILE NOT EOLN DO
BEGIN
  (* input jth char. of predicate on ith transition *)
  READ (T);
  PT(I,J) := T;
  J := J+1
END;
PT(I,J) := '$'; (* add suffix to ith tran. *)
(* display the predicate on ith transition *)
J := 1;
WRITE ('');
WRITE (DATAOUT,'');
WHILE PT(I,J) <> '$' DO
BEGIN
  WRITE (PT(I,J)); (* display on terminal *)
  WRITE (DATAOUT,PT(I,J)); (* display on record *)
  J := J+1
END;
WRITELN;
WRITELN (DATAOUT)
END;

END;

******************************************************************************
(* NAME : SECOND *)
(* FUNCTION : To contain the options of constraints *)
(* definition phase *)
******************************************************************************

PROCEDURE SECOND ;
BEGIN
WRITELN (" *** CONSTRAINTS DEFINITION PHASE *** ");
WRITELN (" < ON CREATE > ");
WRITELN (" * DO YOU WANT TO DEFINE : ");
WRITELN (" * 1. PT --- PREDICATES ON TRANSITIONS ");
WRITELN (" * 2. AT --- ACTIONS AT TRANSITIONS ");
WRITELN (" * 3. PP --- PREDICATES ON PLACES ");
WRITELN (" * 4. AP --- ACTIONS AT PLACES ");
WRITELN (" * ENTER YOUR SELECTION ");
WRITELN (" * PRESS ANY DIGITS OTHER THAN 1, 2, 3, 4 ");
WRITELN (" TO GET OUT OF CONSTRAINTS DEFINITION PHASE ");
END;

(* NAME : RECOM *)
(* FUNCTION : To display the options of constraints *)
(* definition phase on the terminal *)
(* PROCEDURE CALLED : SECOND *)

PROCEDURE RECOM;
BEGIN
  T := 'N';
  WHILE T<>'Y' DO
    BEGIN
      SECOND;
      (* call procedure SECOND *)
      READLN;
      READ(R);
      (* accept user's selection *)
      WRITELN('<',R:1,'>',OK?'--->Y OR N');
      READLN;
      READ(T);
      (* accept user's decision *)
    END;
  END;

(* the main program of procedure CPREDACT *)

BEGIN
  WRITELN(DATAOUT);
  WRITELN(DATAOUT '"** PREDICATE-ACTION DEFINITIONS **"');
  RECOM;
  (* call procedure RECOM *)
  WHILE (R>0) AND (R<5) DO
    BEGIN
      IF R=1 THEN CPREDTR (* call procedure CPREDTR *)
      ELSE IF R=2 THEN CACTTR (* call CACTTR *)
      ELSE IF R=3 THEN CPREDPL (* call *)
      ELSE CACTPL (* call *)
      RECOM
      (* call procedure RECOM *)
    END;
END;

(* NAME : CINITIAL *)
(* FUNCTION : To accept the initial marking M0 *)
(* the initial values of program variables *)
(* and store M0 into array P, assign variables to initial values. *)
(* PROCEDURES CALLED : COMM *)
(* CINITMARK *)
(* CINITVAR *)

PROCEDURE CINITIAL;

(* NAME : CINITMARK *)
(* FUNCTION : To accept the initial marking M0 and *)
(* store it into array P *)

PROCEDURE CINITMARK;
VAR I J : INTEGER;
BEGIN

WRITELN("** ENTER INITIAL MARKING M0 **");
WRITELN("< CREATE OR MODIFY >");
WRITELN;
WRITELN(DATAOUT);
WRITELN(DATAOUT,"** INITIAL MARKING M0 **");
WRITELN(DATAOUT);
/* write the title on terminal and record file */
WRITE("P");
WRITE (DATAOUT,"P");
FOR J:=1 TO L DO
BEGIN
  WRITE("COL",J:1," ");
  WRITE (DATAOUT,"COL",J:1," ");
END;
WRITELN;
WRITELN(DATAOUT);
WRITE (DATAOUT,"");
WRITE ("");
FOR J:=1 TO L DO
BEGIN
  WRITE("-");
  WRITE (DATAOUT,"-");
END;
WRITELN;
WRITELN(DATAOUT);
/* to accept M0 */
FOR I:=1 TO M DO
BEGIN
  /* write 'Pi' on terminal and record file */
  IF I<=9 THEN BEGIN
    WRITELN("P0",I:1);
    WRITE (DATAOUT,"P0",I:"");
  END ELSE BEGIN
    WRITELN("P",I:2);
    WRITE (DATAOUT,"P",I:"");
  END;
READLN;
/* to accept the initial token number in ith place */
FOR J:=1 TO L DO
BEGIN
  /* input token no. in ith place with jth colour */
  READ (R);
  P(I,J) := R
END;
WRITE (" ");
WRITE (DATAOUT," ");
/* display the initial marking on terminal and file */
FOR J:=1 TO L DO
BEGIN
  IF P(I,J)<=9 THEN BEGIN
    WRITE (P(I,J):1," ");
  END;
  IF P(I,J)<>9 THEN BEGIN
    WRITE (P(I,J):1," ");
  END;
END;

- 120 -
WRITE (DATAOUT, P(I,J):1, ")
END
ELSE BEGIN
  WRITE(P(I,J):?, ");
  WRITE (DATAOUT, P(I,J):?, ");
END;

WRITELN (DATAOUT) ;
WRITELN
END;

*******************************************************************************************
(* NAME : CINIVAR *)
(* FUNCTION : To accept initial values of all variables *)
(* and assign them to the variables A,B,...,H *)
(* A1,B1,...,H1, A2,B2,...,H2 *)
*******************************************************************************************
PROCEDURE CINIVAR ;
VAR I J : INTEGER ;
BEGIN
  WRITELN("** ENTER INITIAL VALUES OF VARIABLES **");
  WRITELN;
  IF N1 <> 0 THEN
  BEGIN
    WRITELN("ENTER INTEGER VARIABLE VALUES") ;
    WRITELN("PROGRAM'S USER'S VALUE") ;
    WRITELN("-------------------------") ;
    WRITE ("" ;
    (* accept initial values of integer variables *)
    FOR I:=1 TO N1 DO
    BEGIN
      CH := CHR(I + ORD(\'A\') - 1) ;
      (* display ith program variable name *)
      WRITE(CH, ");
      (* display user's ith variable name on terminal *)
      J := 1;
      WHILE COM(I,J.)<>"$" DO
      BEGIN
        (* display jth char. of ith user's variable name*)
        WRITE (COM(I,J.)) ;
        J := J + 1
      END;
      WRITELN("" ;
      (* input initial values of ith variable *)
      READLN; READ (R) ;
      WRITELN("",R) (* display value *)
    END;
  END;
  IF N<>0 THEN
  BEGIN
    WRITELN("ENTER REAL VARIABLE VALUES") ;
    WRITELN("PROGRAM'S USER'S VALUE") ;
    WRITELN("-------------------------") ;
    WRITE ("" ;
    (* accept all value of real variables *)
  END;
END;
FOR I:=1 TO N2 DO
BEGIN
  CH := CHR(I + ORD(‘A’)) - 1;
  /* display ith program variable name */
  WRITE(CH,’’);
  /* display user’s ith real variable name */
  J := 1;
  WHILE COM1(I,J)<>’$’ DO
BEGIN
  /* write jth character of ith user variable */
  WRITE(COM1(I,J));
  J := J + 1
END;
Writeln(’ ’);
READLN; READ(R); /* input ith variable value */
Writeln(’ ’,R) /* display value */
END
END;
IF N3<>0 THEN
BEGIN
  Writeln(’ ’,< ENTER LOGIC VARIABLE VALUES >’’);
  Writeln(’ ’,PROGRAM'S USER'S VALUE’’);
  Writeln(’ ’,----------------------’’);
  WRITE(’ ’);
  /* accept all initial values of Boolean variables */
  FOR I := 1 TO N3 DO
BEGIN
  CH := CHR(I + ORD(‘A’)) - 1;
  WRITE(CH,’’); /* write ith program variable name */
  /* write user’s ith Boolean variable name */
  J := 1;
  WHILE COM2(I,J)<>’$’ DO
BEGIN
  /* write jth character of ith user variable */
  WRITE(COM2(I,J));
  J := J + 1
END;
Writeln(’ ’);
/* input initial value of ith variable */
READLN; READ(R);
Writeln(’ ’,R) /* display value */
END
END
END
END
END;

(*------------------------------------------------------------------------*)
(* NAME : THIRD *)
(* FUNCTION : TO contain the options of initialization *)
(* marknings and variables phase *)
(*------------------------------------------------------------------------*)
PROCEDURE THIRD ;
BEGIN
  Writeln(’ ’,** ** INITIALIZATION ** **’’);
  Writeln(’ ’,< ON CREATE >’’);
  Writeln(’ ’,** ** *)
BEGIN
  T := "N";
  WHILE T <> "Y" DO BEGIN
    THIRD;
    READLN;
    READ (R);
    IF R = 1 THEN CINIMARK
    ELSE CINIVAR;
    IF R = 1 THEN CINI
    ELSE CINIVAR;
  END;
END;

BEGIN
  COMM;
  WHILE (R>0) AND (R<3) DO
  BEGIN
    IF R = 1 THEN CINIMARK
    ELSE CINIVAR;
  END;
END;

BEGIN
  ZERO;
  MODULE SPECIFICATION
  < ON CREATE >
  DO YOU WANT TO ENTER:
  1. ENTITY DECLARATION PHASE
  2. PREDICATE-ACTION DEFINITION PHASE
END;
```
WRITELN("3. INITIAL MARKING & VARIABLES PHASE");
WRITELN("ENTER YOUR SELECTION");
WRITELN("HIT ANY DIGITS OTHER THAN 1, 2, 3");
WRITELN("TO GET OUT OF MODULE SPECIFICATION SESSION");
END;

(* NAME : INTRO *)
(* FUNCTION : To display the option of module spec. process on terminal *)
(* PROCEDURE CALLED : ZERO *)

PROCEDURE INTRO;
BEGIN
  T := 'N';
  WHILE T <> 'Y' DO
    BEGIN
      ZERO ; (* call procedure ZERO *)
      READLN ;
      READ (R) ; (* accept user's selection *)
      WRITELN ('<''R':1,'''OK? -- Y OR N''');
      READLN ;
      READ (T) (* accept user's decision *)
    END
  END;

(* then main program of procedure CREATE *)

BEGIN
  WRITELN (DATAOUT);
  WRITELN (DATAOUT, '*** PROCESS MODULE SPECIFICATION ***');
  WRITELN (DATAOUT);
  INTRO;
  WHILE (R>0) AND (R<4) DO
    BEGIN
      IF R=1 THEN CDECLARATION (* call *)
      ELSE IF R=2 THEN CPREDACT (* call *)
      ELSE CINITIAL ; (* call *)
      INTRO (* call procedure INTRO *)
    END
  END ;
  CWRITE (* call procedure CWRITE *)
END;

(* NAME : DISPLAY *)
(* FUNCTION : To display all the data of PN on terminal *)
(* and modify these data *)
(* PROCEDURES CALLED : DREAD *)
(* CWRITE *)
(* DINtro *)
(* DDECLARATION *)
(* DPREDACT *)
(* DINITIAL *)
(* PROGRAM IDENTIFIERS : MFLAG -- identify the function of *)
```
(* this procedure: *)
(* MFLAG=0 -- to display data *)
(* MFLAG=1 -- to modify data *)
(* DFLAG -- identify the way of display: *)
(* DFLAG=0 -- <B> potion by potion *)
(* DFLAG=1 -- <A> all together *)
(* FLAG -- indentify whether has already *)
(* called CWRITE so that stored data*)
(* in the output files or not: *)
(* FLAG=0 -- not *)
(* FLAG=1 -- yes *)

DATA RESOURCES:
(* <i> program variables, arrays <when FLAG=1> *)
(* <ii> input files 'DATAIN DATA', DIN DATA' *)
(* <when FLAG=0> *)

*********************************************************************************************
PROCEDURE DISPLAY;
VAR I J K R : INTEGER;
T CH : CHAR;

******************************************************************************
(* NAME: DDECLARATION *)
(* FUNCTION: To display/modify the PN interms of *)
(* total numbers of places, transitions, *)
(* and token colours *)
(* arcs P->T and T->P *)
(* names of variables (integer, real, and *)
(* Boolean). *)

(* PROCEDURES CALLED:
DINTO
DNUMBER
DARC
DINTEGER
DREAL
DLOGIC

******************************************************************************
PROCEDURE DDECLARATION;
VAR L1 : INTEGER;

******************************************************************************
(* NAME: DNUMBER *)
(* FUNCTION: To display/modify the total numbers of *)
(* places, transitions, and colours of PN *)
(* on the terminal: M, N L. *)

(* PROCEDURE CALLED: MOBI

******************************************************************************
PROCEDURE DNUMBER;
VAR V CC;

******************************************************************************
(* NAME: MODI *)
(* FUNCTION: To display and accept user's decision in *)
(* the modify process: *)
(* whether user wants to change current line.*)

******************************************************************************
PROCEDURE MODI;
BEGIN
WRITELN(' * DO YOU WANT TO CHANGE? --- Y OR N');
READLN;
READ (T);  (* input user's decision *)
IF T='Y' THEN
  WRITELN("YOU SHOULD GO BACK TO "MODULE CREATE");
END;
(* the main program of procedure DNUMBER *)
BEGIN
WRITELN("TOTAL # OF PLACES M=M:2"); (* display M*)
IF MFLAG=1 THEN MODI; (* call procedure MODI *)
WRITELN(" TOTAL # OF TRANSITIONS N=N:2");
IF MFLAG=1 THEN MODI; (* call procedure MODI *)
WRITELN(" TOTAL # OF TOKEN TYPES L=L:2");
IF MFLAG=1 THEN MODI; (* call procedure MODI *)
WRITELN
END;

******************************************************************************
(* NAME : DARC *)
(* FUNCTION : To display/modify all the numbers of arcs *)
(* P ---> T and T --> P *)
(* PROCEDURE CALLED : MMODI *)
******************************************************************************
PROCEDURE DARC ;
VAR I J K LAG : INTEGER ;
******************************************************************************
(* NAME : MMODI *)
(* FUNCTION : To change numbers of arcs P->T and T-->P *)
(* in the modify process *)
******************************************************************************
PROCEDURE MMODI ;
VAR I, J, K, PT R1 : INTEGER ;
T1 : CHAR ;
BEGIN
WRITELN("* DO YOU WANT TO CHANGE ? --- Y OR N");
READLN; READ (T); (* accept user's decision *)
PT := 0 ;
WHILE (T = 'Y') AND (PT < 3) DO A
BEGIN
WRITELN("*ENTER THE NEW LINE formed like above :*"); (*
(* read input line like 'P07 T10 0' from terminal*)
(*
READLN; READ (T1); (* read the first cha. *)
WHILE T1 = " " DO READ (T1); (* read blanks *)
IF T1 = "P" THEN PT := 1 (*first letter is 'P'*)
  ELSE IF T1 = "T" THEN PT := 2
    THEN PT := 3 (* it is 'T' *)
  ELSE BEGIN
    (* first non-blank cha. neither 'P' nor 'T' *)
  PT := 3 ;
WRITELN("* INPUT ERROR ENTER AGAIN *")
END;
(* find and read the second non-blank character *)
READ(T1) ;
WHILE T1 = " " DO READ (T1) ;(* found it *)
   (* change character to digit *)
   I := ORD(T1) - ORD("0") ;
   READ (T1) ;(* read the third cha. *)
   I := I * 10 + ORD(T1) - ORD("0") ;(* get digit *)
   (*
   (* find and read the second non-blank cha. group *)
   *)
   READ (T1) ;WHILE T1 = " " DO READ (T1) ;
   READ (T1) ;WHILE T1 = " " DO READ (T1) ;
   J := ORD(T1) - ORD("0") ;(* change to digit *)
   READ (T1) ;(* read the third cha. *)
   J := J * 10 + ORD(T1) - ORD("0") ;(* get integer *)
   (*
   (* find and read the rest non-blank cha. groups *)
   *)
   READ (T1) ;WHILE T1 = " " DO READ (T1) ;
   K := 1 ;
   WHILE K <= L DO
   BEGIN
   R := ORD(T1) - ORD("0") ; READ(T1) ;
   WHILE T1 <> " " DO BEGIN
   R := R * 10 + ORD(T1) - ORD("0") ;
   READ (T1)
   END ;
   (* assign new values to array PARC or TARC *)
   IF PT = 1 THEN PARC(.I,J,K.) := R ;
   IF PT = 2 THEN BEGIN
   R1 := I ; I := J ; J := R1 ;
   TARC(.I J K.) := R
   END ;
   K := K + 1
   END ;
   READLN ;
   WRITELN("*DO YOU WANT TO CHANGE ONE MORE LINE ?");
   WRITELN("----- Y OR N-----");
   READLN ; READ (T)
   END ;
   END ;
   BEGIN
   WRITELN ;
   WRITELN ("* * ARCS P --> T & T --> P * * *");
   WRITELN ;
   (* write the heading for PARC *)
   WRITE ("PLACE TO TRANS.");
   FOR K:= 1 TO L DO
   WRITE ("COL",K:1," ");
   WRITELN ;
   WRITE ("---------------------");
   FOR K:=1 TO L DO
WRITE ('----------') ;
WRITELN ;

(*
  display/modify PARC
(*
(*
FOR I:=1 TO M DO
FOR J:=1 TO N DO
BEGIN
  LAG := 0; (* LAG -- to mark non-zero arcs <LAG=1> *)
  FOR K:=1 TO L DO
    IF PARC(I,J,K.) <> 0 THEN LAG := 1 ;
    (* only display non-zero arcs (*
    IF LAG=1 THEN
      BEGIN
        IF I<=9 THEN WRITE (' P0',I:');
        ELSE WRITE (' P',I:');
        IF J<=9 THEN WRITE (' T0',J:1);
        ELSE WRITE (' T',J:2);
        WRITE(' ');
      FOR K:=1 TO L DO
        WRITE (PARC(I,J,K.):2,' ',)
      WRITE (PARC(I,J,K.):2,' ');
      WRITELN
    END ;
  (* for modifying, call procedure MMODI *)
  IF MFLAG=1 THEN MMODI;
  WRITELN ;
  (* write heading for TARC *)
  WRITE (' TRANS. TO PLACE ');
  (* write the heading *)
  FOR K:=1 TO L DO
    WRITE('COL.',I:1,' ');
  WRITELN ;
  WRITE (' '---------- ');
  FOR K:=1 TO L DO
    WRITE(' '---------- ');
  WRITELN ;
  (*
  modify/modify TARC *
(*
(*
FOR J:=1 TO N DO
FOR I:=1 TO M DO
BEGIN
  LAG := 0; (* LAG -- LAG=1 to mark non-zero arcs *)
  FOR K:=1 TO L DO
    IF TARC(I,J,K.) <> 0 THEN LAG := 1 ;
    (* only display non-zero arcs on terminal *)
    IF LAG=1 THEN
      BEGIN
        IF J<=9 THEN WRITE (' T0',J:1);
        ELSE WRITE (' T',J:2);
        IF I<=9 THEN WRITE (' P0',I:1);
        ELSE WRITE (' P',I:2);
        WRITE(' ');
      END ;
  END ;
FOR K := 1 TO L DO
  WRITE (TARC (.I J K.), 2, " ");
  WRITELN
END

END;

IF MFLAG = 1 THEN MMODI /* for modifying, call MMODI */
END;

******************************************************************************
(* NAME : DMD
(* FUNCTION : To change the name of variables *)
(* INPUT PARAMETER : V -- 8x81 array of char. to represent *)
(* COM, or COM1, or COM2. *)
******************************************************************************

PROCEDURE DMD(VAR V :CC);
  VAR J : INTEGER;
BEGIN
  WRITELN("* DO YOU WANT TO CHANGE THIS NAME? --- Y OR N");
  READLN;
  READ (T);
  IF T = 'Y' THEN
  BEGIN
    WRITELN (" * ENTER NEW NAME OF THIS VARIABLE : ");
    READLN;
    WRITE ("");
    (* input a new name *)
    J := 1;
    WHILE NOT EOLN DO
      BEGIN
        READ (T);
        (* input jth char. of name ll *)
        V (.LL, J.) := T;
        WRITE (V (.L', J.)); (* display new name on terminal *)
        J := J + 1
      END;
    END;
    V (.LL, J.) := ' $' ; (* add suffix to the new name *)
    WRITELN
  END
END;

******************************************************************************
(* NAME : DNUM
(* FUNCTION : To change the total number of any type of *)
(* variables *)
(* INPUT PARAMETER : G -- integer, to represent the total *)
(* number of a type of variables *)
******************************************************************************

PROCEDURE DNUM(VAR G : INTEGER);
BEGIN
  WRITELN("* TOTAL NUMBER OF VARIABLE IN THIS TYPE IS `, G : 3); WRITE (" * ENTER NEW VALUE OF TOTAL NUMBER OF VARIABLES ");
  WRITELN (" IN THIS TYPE ");
  WRITELN(" < IF NO CHANGE HIT 0 > : ");
  READLN;
  READ (R);
  (* input new number *)
  IF R <> 0 THEN G := R
END;

- 129 -
(* NAME : DINTEGER *)
(* FUNCTION : To display or modify the names of user's integer variables (COM) *)
(* PROCEDURES CALLED : DNUM DMOD *)

PROCEDURE DINTEGER ;
VAR I,J : INTEGER ;
BEGIN
  WRITELN ;
  WRITELN (" ** INTEGER VARIABLES **");
  IF MFLAG=1 THEN DNUM (N1); (* call procedure DNUM *)
  (* display integer variable names *)
  WRITELN (" PROGRAM"S USER"S");
  WRITELN (" ----------------------");
  IF N1 = 0 THEN
    WRITELN (" < NIL >")
  ELSE
    FOR I:=1 TO N1 DO
      BEGIN
        CH := CHR(I+ORD('A')-1);
        WRITE (' ',CH:1,'');
        J := 1 ;
        WHILE COM (.I,J.)<>"$" DO
          BEGIN
            WRITE (COM (.I,J));
            J := J+1
          END ;
        WRITELN ;
        (* for modifying, call DMOD to change ith name *)
        IF MFLAG=1 THEN BEGIN
          L1 := I ;
          DMOD (COM) (* call DMOD *)
        END ;
      END ;
  END ;
END ;

(* NAME : DREAL *)
(* FUNCTION : To display or modify names of user's real variables *)
(* PROCEDURES CALLED : DNUM DMOD *)

PROCEDURE DREAL ;
VAR I J : INTEGER ;
BEGIN
  WRITELN ;
  WRITELN (" ** REAL VARIABLES **");
  IF MFLAG=1 THEN DNUM(N2); (* call DNUM *)
  WRITELN (" PROGRAM"S USER"S");
  WRITELN (" ----------------------");
  IF N2 = 0 THEN
    WRITELN (" < NIL >")
ELSE
(* display real variable names on terminal *)
FOR I:=1 TO N2 DO
BEGIN
CH := CHR(I+ORD("A")-1);
WRITE ("",CH:1,"1");
J := 1;
WHILE COM1(.I,J.)<>"$" DO
BEGIN
(* display jth character of ith variable name *)
WRITE(COM1(.I,J.));
J := J+1
END;
WRITELN;
(* for modifying, call DMOD to change ith name *)
IF MFLAG=1 THEN BEGIN
LL := I;
DMOD (COM1)
END
END;
END;

*******************************************************************************
(* NAME : DLOGIC *)
(* FUNCTION : To display or modify names of user's *)
(* Boolean variables *)
(* PROCEDURES CALLED : DNUM *)
(* DMOD *)

*******************************************************************************
PROCEDURE DLOGIC ;
VAR I J : INTEGER ;
BEGIN
WRITELN ;
WRITELN ("* * LOGIC VARIABLES * *");
WRITELN ;
IF MFLAG=1 THEN DNUM(N3) ; (* call DNUM *)
WRITELN ("PROGRAM'S USER'S");
WRITELN (" "-------------------------");
IF N3 = 0 THEN 
WRITELN ("< NIL >") 
ELSE
(* display names of Boolean variables on terminal *)
FOR I:=1 TO N3 DO
BEGIN
CH := CHR (I + ORD("A")-1);
WRITE ("",CH:1,"2");
J := 1;
WHILE COM2(.I,J.)<>"$" DO 
BEGIN
(* display jth character of ith variable name *)
WRITE (COM2(.I,J.)) ;
J := J + 1
END;
WRITELN;
(* for modify, call DMOD to change ith name *)
IF MFLAG=1 THEN BEGIN
   L1 := I ;
   DMOD (COM2)
END

END ;

(*------------------------------------------------------------------*)
(* NAME : DFIRST *)
(* FUNCTION : To contain the options of entity declaration *)
(* declaration phase on DISPLAY or MODIFY *)
(*------------------------------------------------------------------*)

PROCEDURE DFIRST ;
BEGIN
  WRITELN ;
  WRITELN ('* * * ENTITY DECLARATION * * *') ;
  IF MFLAG=0 THEN
    WRITELN ('< ON DISPLAY >') ;
  IF MFLAG=1 THEN
    WRITELN ('< ON MODIFY >') ;
  WRITELN ('------------------------------------------------------------------') ;
  WRITELN ('* DO YOU WANT TO DISPLAY : ');
  WRITELN ('* NUMBERS OF PLACES, TRANSITIONS , ');
  WRITELN ('* AND COLOUR TYPES ');
  WRITELN ('* ARCS P -- T & T --> P ');
  WRITELN ('* 3. INTEGER VARIABLES ');
  WRITELN ('* 4. REAL VARIABLES ');
  WRITELN ('* 5. BOOLEAN VARIABLES ');
  WRITELN ('------------------------------------------------------------------') ;
  WRITELN ('* ENTER YOUR SELECTION *') ;
  WRITELN ('* HIT ANY DIGITS OTHER THAN 1, 2, 3, 4, 5 *') ;
  WRITELN ('* TO GET OUT OF ENTITY DECLARATION SESSION *')
END ;

(*------------------------------------------------------------------*)
(* NAME : DINTO *)
(* FUNCTION : To display the options of entity declaration *)
(* on the terminal and accept user's selection *)
(* PROCEDURE CALLED : DFIRST *)
(*------------------------------------------------------------------*)

PROCEDURE DINTO ;
BEGIN
   T := 'N' ;
   WHILE T<> 'Y' DO
      BEGIN
         DFIRST ;
         READLN ;
         READ (R) ;
         WRITELN ('< , R:1, , > OK ? --- Y OR N') ;
         READLN ;
         READ (T) ;
         END
END ;

(* main program of procedure DDECLARATION *)
(* - 132 - *)
BEGIN
(* in option A, call subprocedures one by one *)
IF DFLAG=1 THEN BEGIN
  DNUMBER;
  DARC;
  DINTEGER;
  DREAL;
  DLOGIC
END ELSE (* in option B, call subprocedures interactively *) BEGIN
DINTO ; (* call DINTO to get user's selection *)
WHILE (R>0) AND (R<6) DO BEGIN
  IF R=1 THEN DNUMBER
  ELSE IF R=2 THEN DARC
  ELSE IF R=3 THEN DINTEGER
  ELSE IF R=4 THEN DREAL
  ELSE DLOGIC ;

DINTO (* call DINTO to get user's next selection *)
END END

(***********************************************************************)
(* NAME : DPREDACT
(* FUNCTION : to display or modify all the predicates *)
(* and actions of PN *)
(* PROCEDURES CALLED : DRECOM *)
(* DPREDTR *)
(* DACTTR *)
(* DPREDPL *)
(* DACTPL *)
(***********************************************************************)
PROCEDURE DPREDACT ;
VAR L2,R : INTEGER ;
T : CHAR ;

(***********************************************************************)
(* NAME : PMOD *)
(* FUNCTION : To modify a predicate or an action *)
(* INPUT PARAMETER : U -- 40x81 array of char. *)
(***********************************************************************)
PROCEDURE PMOD (VAR U : CD ) ;
VAR J : INTEGER ;
BEGIN
  WRITELN ("* DO YOU WANT TO CHANGE ? --- Y OR N");
  READLN ;
  READ (T) ; (* accept user's decision *)
  IF T='Y' THEN BEGIN
WRITELN ('* ENTER THE NEW CONSTRAINT :') ;
READLN ;
(* accept the new constraint *)
J := 1 ;
WRITE (' ') ;
WHILE NOT EOLN DO
BEGIN
(* accept jth character of L2 th constraint *)
READ (T) ;
U(L^,J_) := T ;
WRITE (U(L2,J_)) ; (* display 12th constraint *)
J := J + 1
END ;
WRITELN ;
U(L2,J_) := '$' (* add suffix to 12th cons. *)
END
END ;

********************************************************************************
(* NAME : DPREDTR *)
(* FUNCTION : To display or modify all the predicates *)
(* on all the transitions *)
(* PROCEDURE CALLED : PMOD *)

********************************************************************************
PROCEDURE DPREDTR ;
VAR I,J : INTEGER ;
BEGIN
WRITELN ;
WRITELN ('** PREDICATES ON TRANSITIONS **') ;
WRITELN ('TRANS PREDICATE') ;
WRITELN ('------------------------') ;
FOR I:=1 TO N DO
BEGIN
IF I<=9 THEN WRITE ('TO',I:',')
ELSE WRITE ('T',I:',') ;
(* display the predicate on ith transition *)
J := 1 ;
WHILE PT(I,J) <> '$' DO
BEGIN
WRITE (PT(I,J)) ;
J := J+1
END ;
WRITELN ;
(* for modify, call PMOD to change ith pred. *)
IF MFLAG=1 THEN BEGIN
L2 := I ;
PMOD (PT) (* call PMOD *)
END
END ;

********************************************************************************
(* NAME : DACTTR *)
(* FUNCTION : To display or modify all the actiond at all *)
(* the transitions *)
(* PROCEDURE CALLED : PMOD *)
PROCEDURE DACTTR;
VAR I,J : INTEGER;
BEGIN
WRITELN;
WRITELN("*************** ACTIONS AT TRANSITIONS **********");
WRITELN("TRANS. ACTION");
WRITELN("--------------------------------");
FOR I:=1 TO N DO
BEGIN
IF I<=9 THEN WRITE("T0",I:1," ");
ELSE WRITE("T",I:2," ");
(* display the action at ith transition *)
J := 1;
WHILE AT(.I J ) <> "$" DO
BEGIN
WRITE (AT(.I J ));
J := J + 1
END;
WRITELN;
(* for modify, call PMOD to change ith action *)
IF MFLAG=1 THEN BEGIN
L2 := I;
PMOD(AT) (* call PMOD *)
END
END

 процедура DACTTR
варианты I, J: целые числа;
BEGIN
WRITELN;
WRITELN("*************** ИСХОДЫ ПЕРЕХОДОВ **********");
WRITELN("МЕСТО ПРИБЫТИЯ")
WRITELN("--------------------------------");
FOR I:=1 TO N DO
BEGIN
IF I<=9 THEN WRITE("T0",I:1," ");
ELSE WRITE("T",I:2," ");
(* display the action at ith transition *)
J := 1;
WHILE AT(.I J ) <> "$" DO
BEGIN
WRITE (AT(.I J ));
J := J + 1
END;
WRITELN;
(* for modify, call PMOD to change ith action *)
IF MFLAG=1 THEN BEGIN
L2 := I;
PMOD(AT) (* call PMOD *)
END
END

(* NAME : DPREDOPHA *)
(* FUNCTION : To display or modify all the predicats on places *)
(* on all the places *)
(* PROCEDURE CALLED : PMOD *)

PROCEDURE DPREDOPHA;
VAR I,J : INTEGER;
BEGIN
WRITELN;
WRITELN("*************** ПРЕДИКАТЫ НА МЕСТАХ **********");
WRITELN("МЕСТО ПРЕДИКАТ")
WRITELN("--------------------------------");
FOR I:=1 TO N DO
BEGIN
IF I<=9 THEN WRITE("P0",I:1," ");
ELSE WRITE("P",I:2," ");
(* display the predicate on ith place *)
J := 1;
WHILE PP(.I J ) <> "$" DO
BEGIN
WRITE (PP(.I J ));
J := J + 1
END;
WRITELN;
(* for modify, call PMOD to change ith pred. *)
IF MFLAG=1 THEN BEGIN

- 135 -
L2 := I ;
PMOD(PP)
END

END ;

***************

(* NUME : DACTPL *)
(* FUNCTION : To display or modify all the actions at *)
(* all the places *)
(* PROCEDURES CALLED : PMOD *)

***************

PROCEDURE DACTPL ;
VAR I J : INTEGER ;
BEGIN
WRITELN ;
WRITELN ('** ACTIONS AT PLACES **') ;
WRITELN ('PLACES ACTION') ;
WRITELN ('-------------------') ;
FOR I:=1 TO M DO
BEGIN
IF I<=9 THEN WRITE (' P0',I,'')
ELSE WRITE (' P',I:2,'') ;
(* display ten action at ith place *)
J := 1 ;
WHILE AP(I,J) <> '$' DO
BEGIN
WRITE (AP(I,J)) ;
J := J+1
END ;
WRITELN ;
(* for modify, call PMOD to change ith action *)
IF MFLAG=1 THEN BEGIN
L2 := I ;
PMOD(AP) (* call PMOD *)
END
END ;
WRITELN

***************

(* NAME : DSECOND *)
(* FUNCTION : To contain the options of predicate-action definition phase on DISPLAY or MODIFY *)

***************

PROCEDURE DSECOND ;
BEGIN
WRITELN ;
WRITELN ('** ** PREDICATE-ACTION DEFINITIONS ** **') ;
IF MFLAG=0 THEN
WRITELN('< ON DISPLAY >') ;
IF MFLAG=1 THEN
WRITELN('< ON MODIFY >') ;
WRITELN('** DO YOU WANT TO DISPLAY :') ;
WRITELN('**');
WRITELN('** 1. PT -- PREDICATES ON TRANSITIONS **');
WRITELN('** 2. AT -- ACTIONS AT TRANSITIONS **');
WRITELN('** 3. PP -- PREDICATES ON PLACES **');
WRITELN('** 4. AP -- ACTIONS AT PLACES **');
WRITELN('********************************************************************************');
WRITELN('  * Enter your selection  *');
WRITELN('  * Hit any digits other than 1, 2, 3, 4 *');
WRITELN('  To get out of CONSTRAINTS DEFINITION SESSION*');
END;

(********************************************************************************
(* NAME : DRECOM *)
(* FUNCTION : To display the options on terminal and *)
(* accept user's selections *)
(* PROCEDURE CALLED : DSECOND *)
(********************************************************************************
PROCEDURE DRECOM ;
BEGIN
  T := 'N' ;
  WHILE T <> 'Y' DO
    BEGIN
      DSECOND ; (* call DSECOND to display options *)
      READLN ;
      READ (R) ; (* accept user's selection *)
      WRITELN ('<,R:1,> OK ? -- Y OR N');
      READLN ;
      READ (T) (* accept user's decision *);
    END;
END;

(* Main program of procedure DPREDACT *)
(*
BEGIN
(* in option A, call subprocedures one by one *)
IF DFLAG=1 THEN
  BEGIN
    DPREDTR ;
    DACTTR ;
    DPREDPL ;
    DACTPL
  END
ELSE
  (* In option B, call subprocedures interactively *)
  BEGIN
    DRECOM ; (* call DRECOM to get user's selection *)
    WHILE (R>0) AND (R<5) DO
      BEGIN
        IF R=1 THEN DPREDTR
        ELSE IF R=2 THEN DACTTR
        ELSE IF R=3 THEN DPREDPL
        ELSE DACTPL ;
        DRECOM (* call DRECOM to get next selection *)
      END;
  END
END
END ;
(* NAME : DINITIONAL *)
(* FUNCTION : To display or modify initial marking M0 *)
(* and the initial values of all the variables *)

PROCEDURE DINITIONAL;
VAR I J : INTEGER;
BEGIN
(* write the heading of initial marking M0 *)
  WRITELN;
  WRITELN('** INITIAL MARKING M0 **');
  IF MFLAG=0 THEN
    WRITELN('< DISPLAY >');
  IF MFLAG=1 THEN
    WRITELN('< MODIFY >');
  WRITE('PLACE');
  FOR I:=1 TO L DO
    WRITE('COL','I:',' ');
  WRITELN;
  WRITE('-----');
  FOR I:=1 TO L DO
    WRITE('------');
  WRITELN;
(* display initial marking M0 *)
  FOR I:=1 TO M DO
  BEGIN
    IF I<=9 THEN WRITE(' P0','I:',' '); ELSE WRITE(' P','I:',' '); FOR J:=1 TO L DO
      WRITE(P(I,J):2,' '); WRITELN;
(* for modify, change M0 *)
  IF MFLAG=1 THEN
  BEGIN
    WRITE('DO YOU WANT TO CHANGE THIS LINE ?');
    WRITELN(' Y OR N');
    READLN;
    READ(T); (* accept user's decision *)
    (* accept token num. in ith place for each colour *)
    IF T='Y' THEN
    BEGIN
      WRITELN('ENTER NEW VALUE OF THIS LINE :');
      READLN;
      WRITE(' '); FOR J:=1 TO L DO
        BEGIN
          READ (R);
          P(I,J) := R;
          WRITE(P(I,J):2,'');
        END;
      WRITELN;
    END;
  END;
END;
END;
IF N1 <> 0 THEN
BEGIN
  (* write heading for initial values of integer variables *)
  WRITELN('** INTEGER VARIABLE VALUES **');
  IF MFLAG=0 THEN
    WRITELN('< DISPLAY >');
  IF MFLAG=1 THEN
    WRITELN('< MODIFY >');
  WRITELN('PROGRAM'S USER'S VALUES');
  WRITELN('--------------------------------');
  FOR I:=1 TO N1 DO
    BEGIN
      (* display program name of ith integer variables *)
      WRITE('CH := CHR ( I + ORD('A') - 1 );
      WRITE(CH,'');
      (* display user's name of ith integer variable *)
      J := 1;
      WHILE COM(.I,J.)<>"$" DO
        BEGIN
          WRITE(COM(.I,J));
        J := J + 1
        END;
      (* display initial value of ith integer variables *)
      CASE I-1 OF
        0 : WRITELN(A); 1 : WRITELN(B);
        2 : WRITELN(C); 3 : WRITELN(D);
        4 : WRITELN(E); 5 : WRITELN(F);
        6 : WRITELN(G); 7 : WRITELN(H)
      END;
    END;
  END;
END;
IF N2 <> 0 THEN
BEGIN
  (* write heading for initial values of real var. *)
  WRITELN('** REAL VARIABLE VALUES **');
  IF MFLAG=0 THEN
    WRITELN('< DISPLAY >');
  IF MFLAG=1 THEN
    WRITELN('< MODIFY >');
  WRITELN('PROGRAM'S USER'S VALUE');
  WRITELN('--------------------------------');
  WRITELN('');
  FOR I:=1 TO N2 DO
    BEGIN
      (* display program name of ith real variable *)
      CH := CHR ( I + ORD('A') - 1 );
      WRITE(CH,'');
      (* display user's name of ith real variable *)
      J := 1;
      WHILE COM1(.I,J.)<>"$" DO.
        BEGIN
        WRITE(COM1(.I,J));
        J := J + 1
      END;
    END;
END;
END;

(* display initial value of ith real variable *)
CASE I-1 OF

0: WRITELN(A1); 1: WRITELN(B1);
2: WRITELN('C1'); 3: WRITELN(D1);
4: WRITELN(E1); 5: WRITELN(F1);
6: WRITELN(G1); 7: WRITELN(H1);
END
END
END;

IF N3 <> 0 THEN
BEGIN
(* write heading for initial Boolean variables *)
WRITELN(" ** LOGIC VARIABLE VALUES **");
IF MFLAG=0 THEN
WRITELN(" < DISPLAY >");
IF MFLAG=1 THEN
WRITELN(" < MODIFY >");
WRITELN(" PROGRAM'S USER'S VALUE");
WRITELN(" -------------------------");
WRITE ("");
FOR I:=1 TO N3 DO
BEGIN
(* display program name of ith Boolean var. *)
CH := CHR(I + ORD('A') - 1);
WRITE(CH);
(* display user's name of ith Boolean var. *)
J := 1;
WHILE COM2(.I,J.)<>"$" DO BEGIN
WRITE(COM2(.I,J.));
J := J + 1
END;
(* display initial value of ith Boolean var. *)
WRITE(" ");
CASE I-1 OF
0: WRITELN(A2}; 1: WRITELN(B2);
2: WRITELN(C2); 3: WRITELN(D2);
4: WRITELN(E2); 5: WRITELN(F2);
6: WRITELN(G2); 7: WRITELN(H2);
END
END
END;

(************************************************************************************
(* NAME : DZERO *)
(* FUNCTION : to contain to options Module DISPLAY *)
(* and Module MODIFY *)
************************************************************************************)

PROCEDURE DZERO;
BEGIN
WRITELN;
IF MFLAG=0 THEN
WRITELN ('*** MODULE DISPLAY SESSION ***');
IF MFLAG=1 THEN
WRITELN ('*** MODULE MODIFY SESSION ***');
WRITELN ('*******************************');
WRITELN ('* DO YOU WANT TO INPUT *');
WRITELN ('1. ENTITY DECLARATION *');
WRITELN ('2. PREDICATE-ACTION DEFINITIONS *');
WRITELN ('3. INITIAL MARKING M0 *');
WRITELN ('*******************************');
WRITELN ('* ENTER YOUR SELECTION *');
WRITE ('* HIT ANY DIGITS OTHER THAN 1, 2, 3 *');
WRITELN ('* TO GET OUT OF DISPLAY SESSION *');
END ;

(******************************************************************)
(* NAME : DINTRO *)
(* FUNCTION : To display the options on terminal and *)
(* accept user's selection *)
(* PROCEDURE CALLED : DZERO *)
(******************************************************************)

PROCEDURE DINTRO ;
BEGIN
  T := 'N' ;
  WHILE T<> 'Y' DO
    BEGIN
      DZERO ; /* call DZERO to display options */
      READLN ;
      READ (R) ; /* accept user's selection */
      WRITELN ('<,R:1,> OK ? --- Y OR N') ;
      READLN ;
      READ (T) /* accept user's decision */
    END
  END ;

(******************************************************************)
(* NAME : DREAD *)
(* FUNCTION : To read data from input files: DATAIN DATA *)
(* and DIN DATA *)
(******************************************************************)

PROCEDURE DREAD ;
VAR I J K : INTEGER ;
BEGIN
  /* read total num. of places, transitions, and colours */
  READ (DATAIN,M) ;
  READ (DATAIN,N) ;
  READLN (DATAIN,L) ;
  /* read arcs from places to transitions */
  FOR I:=1 TO M DO
    FOR J:=1 TO N DO
      BEGIN
        FOR K:=1 TO L DO
          READ (DATAIN PARC(.I,J,K)) ;
          READLN (DATAIN)
      END ;
  /* read arcs from transitions to places */
  FOR J:=1 TO N DO
    ...
FOR I:=1 TO M DO
BEGIN
FOR K:=1 TO L DO
READ (DATAIN TARC(.I,J,k.));
READLN (DATAIN)
END;
(* read initial marking M0 *)
FOR I:=1 TO M DO
BEGIN
FOR K:=1 TO L DO
READ (DATAIN,P(.I,K.));
READLN (DATAIN)
END;
(* read total num. of integer, real, Boolean variables *)
READLN (DATAIN,N1,N2,N3);
(* read initial values of integer variables *)
IF N1 <> 0 THEN
FOR I:=1 TO N1 DO
CASE I-1 OF
  0 : READLN (DATAIN,A);
  1 : READLN (DATAIN,B);
  2 : READLN (DATAIN,C);
  3 : READLN (DATAIN,D);
  4 : READLN (DATAIN,E);
  5 : READLN (DATAIN,F);
  6 : READLN (DATAIN,G);
  7 : READLN (DATAIN,H);
END;
(* read initial values of real variables *)
IF N2 <> 0 THEN
FOR I:=1 TO N2 DO
CASE I-1 OF
  0 : READLN (DATAIN,A1);
  1 : READLN (DATAIN,B1);
  2 : READLN (DATAIN,C1);
  3 : READLN (DATAIN,D1);
  4 : READLN (DATAIN,E1);
  5 : READLN (DATAIN,F1);
  6 : READLN (DATAIN,G1);
  7 : READLN (DATAIN,H1);
END;
(* read user's names of integer variables *)
WHILE NOT EOF (DIN) DO
BEGIN
FOR I:=1 TO 8 DO
BEGIN
  J:=1; READ (DIN,T);
  WHILE T <> "$" DO
  BEGIN
    COM (.I,J) := T;
    J := J+1; READ (DIN,T)
  END;
  READLN (DIN);
  COM (.I,J.) := "$";
END;
(* read user's names of real variables *)
FOR I:=1 TO 8 DO
BEGIN
  J:=1; READ (DIN,T);
  WHILE T <> "$" DO
  BEGIN
    COM1 (.I,J) := T;
  END;
\[ \text{J := J+1 ; READ (DIN,T)} \]
\[ \text{END} ; \]
\[ \text{READLN (DIN)} ; \]
\[ \text{COM1(.I J.) := "$"} \]
\[ \text{END} ; \]

(* read user's name of Boolean variables *)

\[ \text{FOR I:=1 TO 8 DO} \]
\[ \text{BEGIN} \]
\[ \text{J := 1 ; READ (DIN,T)} ; \]
\[ \text{WHILE T <> "$" DO} \]
\[ \text{BEGIN} \]
\[ \text{COM2(.I,J.) := T ;} \]
\[ \text{J := J+1 ; READ (DIN,T)} \]
\[ \text{END} ; \]
\[ \text{READLN (DIN)} ; \]
\[ \text{COM2(.I,J.) := "$"} \]
\[ \text{END} ; \]

(* read all predicates on transitions *)

\[ \text{FOR I:=1 TO N DO} \]
\[ \text{BEGIN} \]
\[ \text{J := 1 ; READ (DIN,T)} ; \]
\[ \text{WHILE T <> "$" DO} \]
\[ \text{BEGIN} \]
\[ \text{PT(.I,J.) := T ;} \]
\[ \text{J := J+1 ; READ (DIN,T)} \]
\[ \text{END} ; \]
\[ \text{READLN (DIN)} ; \]
\[ \text{PT(.I,J.) := "$"} \]
\[ \text{END} ; \]

(* read all actions at transitions *)

\[ \text{FOR I:=1 TO N DO} \]
\[ \text{BEGIN} \]
\[ \text{J := 1 ; READ (DIN,T)} ; \]
\[ \text{WHILE T <> "$" DO} \]
\[ \text{BEGIN} \]
\[ \text{AT(.I,J.) := T ;} \]
\[ \text{J := J+1 ; READ (DIN,T)} \]
\[ \text{END} ; \]
\[ \text{READLN (DIN)} ; \]
\[ \text{AT(.I,J.) := "$"} \]
\[ \text{END} ; \]

(* read all predicates on places *)

\[ \text{FOR I:=1 TO M DO} \]
\[ \text{BEGIN} \]
\[ \text{J := 1 ; READ(DIN,T)} ; \]
\[ \text{WHILE T <> "$" DO} \]
\[ \text{BEGIN} \]
\[ \text{PP(.I J.) := T ;} \]
\[ \text{J := J+1 ; READ(DIN,T)} \]
\[ \text{END} ; \]
\[ \text{READLN (DIN)} ; \]
\[ \text{PP(.I,J.) := "$"} \]
\[ \text{END} ; \]

(* read all action at places *)
FOR I := 1 TO M DO
BEGIN
  J := 1; READ (DIN,T);
  WHILE T <> "$" DO
    BEGIN
      AP(I,J) := T;
      J := J + 1; READ (DIN,T);
    END;
    READLN (DIN);
    AP(I,J) := "$"
  END;
END;

(* the main program of procedure DISPLAY *)
BEGIN
  IF FLAG = 0 THEN BEGIN
    DREAD;
    CWRITE
    END;
    WRITELN(' WHICH DISPLAY FORM DO YOU WANT TO CHOOSE : ');
    WRITELN(' A  DISPLAY ALL THE INFORMATION AT ONCE');
    WRITELN(' B. DISPLAY INFORMATION INTERACTIVELY');
    READLN;
    READ(T);
    IF T = "A" THEN
      (* in option A call subprocedures one by one *)
      BEGIN
        DFLAG := 1; (* set DFLAG=1 to mark option A *)
        DDECLARATION;
        DPREDACT;
        DINITIAL;
        DFLAG := 0 (* initial DFLAG *)
      END
      (* in option B, call subprocedures interactively *)
      ELSE IF T="B" THEN
      BEGIN
        DINTRO; (* call DINTRO to get user's selection *)
        WHILE (R>0) AND (R<4) DO
          BEGIN
            IF R=1 THEN DDECLARATION
            ELSE IF R=2 THEN DPREDACT
            ELSE DINITIAL;
          END
        DINTRO
      END
    END
END;

(**********************************************************************)
(* NAME : NEW *)
(* FUNCTION : To contain the options of the package *)
(**********************************************************************)
PROCEDURE NEW;
BEGIN
PROCEDURE NINTO;
BEGIN

T := 'N' ;
WHILE T <> 'Y' DO
BEGIN
NEW ;
READLN ;
READ (R) ;
WRITELN (' <"R:1,"> OK ? --- Y OR N') ;
READLN ;
READ (T)
END
END ;

(* the main program of this package *)
BEGIN
(* open the disk files *)
RESET (DATAIN) ;
RESET (DIN) ;
REWRITE (DATAOUT) ;
REWRITE (OUTIN) ;
REWRITE (OUT) ;
(* initialize the program identifiers *)
FLAG := 0 ;
DFLAG := 0 ;
MFLAG := 0 ;
(* start process, call procedures *)
NINTO ;
WHILE (R>0) AND (R<5) DO
BEGIN
IF R=1 THEN CREATE
ELSE IF R=2 THEN DISPLAY
ELSE IF R=3 THEN BEGIN
MFLAG := 1 ;
DISPLAY ;
END
END

- 145 -
CWRITE ;
MFLAG := 0
END
ELSE ANALYSIS (P,PARC,TARC,PP,
PT AP,AT,M,N,L) ;
(*$E+$*)

PROGRAM PACK (INPUT/, OUTPUT, DATAIN,DATAOUT,OUTIN,DIN,OUT) ;

TYPE LIST  = PACKED ARRAY (.1..40, 1..9.) OF INTEGER ;
CA  = PACKED ARRAY (.1..40, 1..40.) OF INTEGER ;
CB  = PACKED ARRAY (.1..40, 1..40 , 1..9.) OF INTEGER ;
CC  = PACKED ARRAY (.1..8, 1..81.) OF CHAR ;
CD  = PACKED ARRAY (.1..40, 1..81.) OF CHAR ;
CF  = PACKED ARRAY (.1..40 ) OF INTEGER ;

VAR PARC, TARC : CB ;
P : LIST ;
COM, COM1, COM2 : CC ;
PP, PT, AP, AT : CD ;
N ,M,L,N1,N2,N3 : INTEGER ;
A B ,C,D,E F G,H : INTEGER ;
A^,B1,C1,D1,E1,F1,G1,H1 : REAL ;
A2,B2,C2,D2,E2,F2,G2,H2 : BOOLEAN ;
T : CHAR ;
R FLAG, DFLAG, MFLAG : INTEGER ;
DATAIN DATAOUT, OUTIN : TEXT ;
DIN.OUT : TEXT ;

******************************************************************************

(* NAME : ANALYSIS *)
(* FUNCTION : To analyze the numerative and incidence *)
(* properties of the PN *)
(* INPUT PARAMETERS : *)
(* P -- MxL array, initial marking M0 *)
(* PARC -- MxNxl array, arcs P->T *)
(* TARC -- MxNxl array, arcs T->P *)
(* PP -- Mx80 array, predicates on places *)
(* PT -- Nx80 array, pred. on transitions *)
(* AP -- Mx80 array, actions at places *)
(* AT -- N x80 array, actions at trans. *)
(* M -- integer, total number of places *)
(* N -- integer, total number of trans. *)
(* L -- integer, total number of colours *)
(* PROCEDURES CALLED : AINTO *)
(* REACHABILITY *)
(* INCIDENCE *)

******************************************************************************

PROCEDURE ANALYSIS (VAR P:LIST; VAR PARC,TARC :CB; VAR PP,PT AP,AT:CD; VAR M,N,L : INTEGER );

******************************************************************************

(* NAME : REACHABILITY *)
(* FUNCTION : To analyze the numerative properties of PN *)
(* PROGRAM CONSTANTS : *)
(* E0 -- integer, the max. levels allowed of the tree E0=15 *)
(* E4 -- integer, E4 = E0 + 1 = 16 *)
(* PROGRAM VARIABLES : *)
(* K1 -- integer, the current level # of the tree *)
(* K0 -- integer, the new marking Mk1+1 = Mk0 *)
(* K2 -- integer, rotation var. for finding Mk0 *)
(* K3 -- integer, if Mk1+1 = Mk0, let K3 = K1+1 *)
(* K4 -- integer, if Mk1+1 >=Mk2, let K4 = K1+1 (w-symbol) *)
(* L1 -- integer, sum of unfirable Tj in level K1 *)
(* L2 -- integer, sum of such levels which has only one *)
(* one branch (levellock) *)

(* PROGRAM IDENTIGIERS *)
(* SAF -- 0/1, the PN is safe / not *)
(* CFLAG -- 0/1, the PN is conservative / not *)
(* DDLK -- 1/0, the PN has deadlock / not *)
(* LFLK -- 1/0, the PN has livelock / not *)
(* LVLP -- 1/0, the PN has live-loop / not *)
(* HMST -- 1/0, M0 is home-state / not *)
(* KFOUND -- 1/0, found K0 so that Mk+1 = Mk0 / not *)
(* WFLAG -- 1/0, the PN has predicates or actions/not *)
(* E3 -- 1/0, the current level > E0 / not *)
(* GG -- 1/0, during the analysis the user wants to *)
(* give variable value(s) / not *)

(* PROGRAM ARRAYS *)
(* MR'k1,i,k -- E0xMxL, integer, marking of level k1 (Mk1) *)
(* TT(k1,j) -- E0xN, 1/0, at level k1, Tj is firable/not *)
(* BD(k) -- 1<=k<=L, max Mkl(i,k) for all 1<=i<=M *)
(* CS(k) -- 1<=k<=L, sum of M0(i,k) for all 1<=i<=M *)
(* CSK(k) -- 1<=k<=L, sum of Mk1(i,k) for all 1<=i<=M *)
(* Q(k1) -- 1<=k1<=E0, # of transition fired in level k1 *)
(* Ql(k1) -- 1<=k1<=E0, the # of marking in level k1 *)
(* QI 'k1,i) -- E4x8, integer, the values of A,B,...H *)
(* QR'k1,i) -- E4x8, real, the values of A1,B1,...H1 *)
(* QL'k1,i) -- E4x8, Boolean, the values of A2,B2,...H2 *)

(* PROCEDURES CALLED *)
(* PREP *)
(* RECEIVE *)
(* CHECK *)
(* FIND *)
(* FIRING *)
(* LEAF *)
(* LOCK *)
(* VARIABLES *)

(*PROCEDURE REACHABILITY *)
LABEL 100, 200, 300 ;
CONST E0 = 15 ; E4=16 ;
VAR L1 L2, K0, K1, K2, K3, K4, R, R',E3 : INTEGER ;
GG, SAF, FLAG, CFLAG, FOUND,KFOUND,WFLAG : INTEGER ;
DDLK, LVLP, HMST : INTEGER ;
Q, QI : ARRAY(.1.1E4,1.8.) OF INTEGER ;
QR : ARRAY(.1.1E4,1.8.) OF REAL ;
QL : ARRAY(.1.1E4,1.8.) OF BOOLEAN ;
BD : ARRAY(.1.9.) OF INTEGER ;
QK : ARRAY(.1.40,1.9.) OF INTEGER ;
CS, CSK : ARRAY(.1.9.) OF INTEGER ;
TT : ARRAY(.1.1E4,1.40.) OF INTEGER ;
MR : ARRAY(.1.1E4,1.40,1.40.) OF INTEGER ;
I, J, K, XX, YY : INTEGER ;
BL : BOOLEAN ;
R0 : REAL ;
***************
(* NAME : PREP *)
(* FUNCTION : To initialize all the variables and arrays *)
(* and ask whether the user wants to input *)
(* new values of PN variables during *)
(* reachability analysis or not *)

***************

PROCEDURE PREP;
VAR I, K : INTEGER;
BEGIN
WRITELN ('PREP');
(* initialize marking *)
FOR K := 1 TO L DO
  FOR I := 1 TO M DO
    MR(.I, I, K) := P(.I, K);
(* initialize program variables *)
SAF := 0; FLAG := 0; CFLAG := 0; FOUND := 0;
KFOUND := 0; HMST := 0;
E3 := 0; GG := 0; DLLK := 0; LVK := 0; LVLP := 0;
WFLAG := 0; K4 := 0;
(* initial some arrays *)
FOR K := 1 TO L DO
  BEGIN
    BD(.K.) := 0;
    CS(.K.) := 0;
    CSK(.K.) := 0
  END;
FOR K := 1 TO L DO
  FOR I := 1 TO M DO
    QK(.I, K) := 0;
(* set WFLAG to 1 when there are some PN variables *)
  I := 1;
  WHILE (I <= M) AND (WFLAG=0) DO
    IF (PP(.I, 1, 1) <> \$') OR (AP(.I, 1, 1) <> \$')
      THEN WFLAG := 1
      ELSE I := I + 1;
  I := 1;
  WHILE (I <= N) AND (WFLAG=0) DO
    IF (PT(.I, 1, 1) <> \$') OR (AT(.I, 1, 1) <> \$')
      THEN WFLAG := 1
      ELSE I := I + 1;
(* get CS for determining conservation *)
  END;
FOR K := 1 TO L DO
  FOR I := 1 TO M DO
    CS(.K.) := CS'.K.) + P(.I, K);
(* if there are PN variables, *)
(* ask does the user wants input new values or not *)
  IF WFLAG = 1 THEN
    BEGIN
      WRITELN ('* DO YOU WANT GIVE VARIABLE VALUES FROM TERMINAL?');
      WRITELN ('DURING ANALYSISNG ? --- Y OR N *');
      READLN;
      READ (T);
(* accept user's decision *)
      IF T = 'Y' THEN GG := 1;(* set GG=1 to indicate 'Y' *)
    END;
END.
END;
(* display M0 -- MR(.1,i,k) *)
WRITE (' M<1,0> : < ' );
FOR K:= 1 TO E4 DO
    Q1(K) := 0;
FOR K:=1 TO L DO
END
END;

***********************************************************************
(* NAME : RECEIVE *)
(* FUNCTION : To receive PN variable values from terminal *)
(* PROGRAM ARRAYS : *)
(* SS(i) -- 1<=i<=81, char. store input string *)
(* RR'i) -- 1<=i<=81, integer, store input values *)
(* DATA RESOURCE : from terminal *)
***********************************************************************

PROCEDURE RECEIVE;
VAR I,J,J1 : INTEGER;
SS : ARRAY(.1..81.) OF CHAR;
RR : ARRAY(.1..81.) OF INTEGER;
T0 : CHAR;
BEGIN
    WRITELN (' RECEIVE');
    REPEAT
    (* give the user input instruction *)
    WRITELN (' ENTER NEW VALUES (IF ANY)TO THE VARIABLE(S):');
    WRITELN (' A, B,... < E G. A=1 >. IF NO ENTRY PRESS N');
    READLN ; READ (T0);
    (* accept a new value *)
    (* assign the new value to the corresponding PN var. *)
    IF T0 <> 'N' THEN
        BEGIN
            SS(.1.) := T0;
            I := 2;
            WHILE NOT EOLN DO
                BEGIN
                    READ (T);
                    SS(.I.) := T;
                    I := I + 1
                END;
        END;
    IF SS(.2.) = '=' THEN BEGIN
        (* for integer variable *)
        R := ORD(SS.'3.') - ORD('0'); (*the value*)
        R11:=ORD(SS.'1.') - ORD('A'); (*# of var.*)
        CASE R11 OF
            0 : A := R;
            1 : B := R;
            2 : C := R;
            3 : D := R;
    END
4: \ E := R;  \ 5: \ F := R;  \\
6: \ G := R;  \ 7: \ H := R
END

ELSE IF SS(.2.) = '1'
THEN BEGIN
(* for real variable *)
FOR J := 4 TO I-1 DO
  IF SS(.J ) = 'I'.
    THEN J' := J
ELSE RR(.J.) := ORD(SS(.I .)) - ORD('0');;
R0 := 0.0 ;
FOR J := 4 TO J1 - 1 DO
  R0 := R0 * 10 + RR(.J .);
FOR J := J' + 1 TO I-1 DO
  R0 := R0 + RR(.J .) * EXP((J-J1) * LN(0.1)) ;
R1 := ORD(SS(.1.)) - ORD('A');
CASE R1 OF
  0: \ A1 := R0;  \ 1: \ B1 := R0;
  2: \ C1 := R0;  \ 3: \ D1 := R0;
  4: \ E1 := R0;  \ 5: \ F1 := R0;
  6: \ G1 := R0;  \ 7: \ H1 := R0
END
END

ELSE IF SS(.2.) = '2'
THEN BEGIN
(* for Boolean variable *)
IF SS(.4.) = 'T' THEN BL := TRUE
  ELSE BL := FALSE;
R1 := ORD(SS(.1.)) - ORD('A');
CASE R1 OF
  0: \ A2 := BL;  \ 1: \ B2 := BL;
  2: \ C2 := BL;  \ 3: \ D2 := BL;
  4: \ E2 := BL;  \ 5: \ F2 := BL;
  6: \ G2 := BL;  \ 7: \ H2 := BL
END
END

ELSE BEGIN
(* for error of input form *)
  WRITELN ('* SORRY WHAT YOU ENTERED IS NOT CORRECT FORMED');
  WRITELN ('* --- ENTER IT AGAIN *');
END
UNTIL T0 = 'N'
END
PROCEDURE CHECK;
VAR I,J,K : INTEGER;
BEGIN
  WRITELN ("CHECK");
  (* check safeness *)
  I := 1;
  WHILE (I<=M) AND (SAF=0) DO
    BEGIN
      K := 1;
      WHILE (K<=L) AND (SAF=0) DO
        IF MR(.K1,I,K.) > 1 THEN SAF := 1 (* not safe *)
          ELSE K := K + 1;
      I := I + 1;
    END;
  (* get max. value of BD for checking boundedness *)
  FOR K:=1 TO L DO
    FOR I:=1 TO M DO
      IF MR(.K1,I,K.) > BD(.K.) THEN BD(.K.) := MR(.K1,I,K.);
      (* check conservation *)
    FOR K:=1 TO L DO CSK(.K.) := 0;
    K := 1;
    WHILE (K<=L) AND (CFLAG=0) DO
      BEGIN
        FOR I:=1 TO M DO
          CSK(.K.) := MR(.K1,I,K.) + CSK(.K.) ;
        IF CSK(.K.) <> CS(.K.) THEN CFLAG := 1
          ELSE K := K + 1
      END
END;

PROCEDURE FIND;
VAR I,J,K,I1,J1 : INTEGER;
BEGIN
  WRITELN ("FIND");
  FOR J:=1 TO N DO
    BEGIN
      (* check Tj is enabled or not *)
      (* *)
      FLAG := 0;
      K := 1;
      WHILE (K<=L) AND (FLAG=0) DO
        BEGIN
          I := 1;
          WHILE (I<=M) AND (FLAG=0) DO
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (* *)
            (*)}
IF MR(.KL,I,K.) < PARC(.I,J K.) THEN FLAG := 1
ELSE I := I + 1;
K := K + 1
END;

(* check the predicate associated with Tj. *)
(* is true or not *)
IF PT(.J,1.) <> "$" THEN
IF FLAG = 0 THEN
IF PT(.J,2.) = "$" THEN BEGIN
(* for the predicate with integer PN var. *)
R1 := ORD(PT'.J,1.) - ORD("A");
R := 0; I1 := 3;
WHILE PT(.J I1.) = " " DO I1 := I1 + 1;
WHILE (PT(.J,II.) <> "A") AND
(PT(.J,II.) <> "$") DO BEGIN
R := R * 10 + ORD(PT(.J,II.)) - ORD("0");
II := II + 1
END;
CASE R1 OF
0 : IF A<>R THEN FLAG := 1;
1 : IF B<>R THEN FLAG := 1;
2 : IF C<>R THEN FLAG := 1;
3 : IF D<>R THEN FLAG := 1;
4 : IF E<>R THEN FLAG := 1;
5 : IF F<>R THEN FLAG := 1;
6 : IF G<>R THEN FLAG := 1;
7 : IF H<>R THEN FLAG := 1
END;
IF PT(.J,II.) = "A" THEN BEGIN
(* for the predicate with form "A=1 and B=0" *)
II := II + 3;
WHILE PT'.J,II.)=" " DO I1 := I1 + 1;
R1 := ORD(PT'.J II.) - ORD("A");
II := II + 2;
WHILE PT'.J,II.) <> "$" DO BEGIN
R := R * 10 + ORD(PT'.J,II.) - ORD("0");
II := II + 1
END;
CASE R1 OF
0 : IF A <> R THEN FLAG := 1;
1 : IF B <> R THEN FLAG := 1;
2 : IF C <> R THEN FLAG := 1;
3 : IF D <> R THEN FLAG := 1;
4 : IF E <> R THEN FLAG := 1;
5 : IF F <> R THEN FLAG := 1;
6 : IF G <> R THEN FLAG := 1;
7 : IF H <> R THEN FLAG := 1
END
END

- 153 -
ELSE IF PT(.J 2.)='1'
THEN BEGIN

(*for predicate with real PN var.*)
I' := 4 ;
R0 := 0.0 ;
WHILE PT('J,1l.) <> '1' DO
BEGIN
    R := ORD(PT('J,1l.)) - ORD('0');
    R0 := R0 * 10 + R ;
    I1 := I1 + 1
END;
J1 := I1 ;
I1 := J1 + 1 ;
WHILE PT('J,1l.) <> 'S' DO
BEGIN
    R := ORD(PT('J,1l.)) - ORD('1');
    R0 := R0 + EXP((I1-J1) * LN(0.1));
    I1 := I1 + 1
END;
R1 := ORD(PT('J,1.)) - ORD('A');
CASE R' OF
  0 : IF A1<>R0 THEN FLAG := 1 ;
  1 : IF B1<>R0 THEN FLAG := 1 ;
  2 : IF C1<>R0 THEN FLAG := 1 ;
  3 : IF D1<>R0 THEN FLAG := 1 ;
  4 : IF E1<>R0 THEN FLAG := 1 ;
  5 : IF F1<>R0 THEN FLAG := 1 ;
  6 : IF G1<>R0 THEN FLAG := 1 ;
  7 : IF H1<>R0 THEN FLAG := 1
END
ELSE BEGIN

(*for predicate with Boolean PN var.*)
IF PT('J,4.)='T' THEN BL := TRUE
ELSE BL := FALSE ;
R1 := ORD(PT('J,1.)) - ORD('A');
CASE R' OF
  0 : IF A2<>BL THEN FLAG := 1 ;
  1 : IF B2<>BL THEN FLAG := 1 ;
  2 : IF C2<>BL THEN FLAG := 1 ;
  3 : IF D2<>BL THEN FLAG := 1 ;
  4 : IF E2<>BL THEN FLAG := 1 ;
  5 : IF F2<>BL THEN FLAG := 1 ;
  6 : IF G2<>BL THEN FLAG := 1 ;
  7 : IF H2<>BL THEN FLAG := 1
END
END ;

I := 1 ;

(*
check the predicate at Tj input places
(*

WHILE (FLAG=0) AND (I<=M) DO
BEGIN
FOUND := 0;
K := 1;
WHILE (K <= L AND (FOUND = 0)) DO
  IF (PARC(I, J, K) <> 0) OR (TARC(I, J, K) <> 0)
    THEN FOUND := 1
  ELSE K := K + 1;
  IF PP(I, L) <> "$" THEN
    IF FOUND = 1 THEN
      IF PP(I, 2) = "1"
        THEN BEGIN
          (* for predicate with integer var. *)
          R := ORD(PP(I, 3)) - ORD("0");
          R1 := ORD(PP(I, L)) - ORD("A");
          CASE R1 OF
            0 : IF A < R THEN FLAG := 1;
            1 : IF B < R THEN FLAG := 1;
            2 : IF C < R THEN FLAG := 1;
            3 : IF D < R THEN FLAG := 1;
            4 : IF E < R THEN FLAG := 1;
            5 : IF F < R THEN FLAG := 1;
            6 : IF G < R THEN FLAG := 1;
            7 : IF H < R THEN FLAG := 1
          END;
        END
      ELSE IF PP(I, 2) = "1"
        THEN BEGIN
          (* pred. with real var. *)
          R := ORD(PP(I, 4)) - ORD("0");
          R1 := ORD(PP(I, L)) - ORD("A");
          CASE R1 OF
            0 : IF A1 < R THEN FLAG := 1;
            1 : IF B1 < R THEN FLAG := 1;
            2 : IF C1 < R THEN FLAG := 1;
            3 : IF D1 < R THEN FLAG := 1;
            4 : IF E1 < R THEN FLAG := 1;
            5 : IF F1 < R THEN FLAG := 1;
            6 : IF G1 < R THEN FLAG := 1;
            7 : IF H1 < R THEN FLAG := 1
          END;
        END
      ELSE BEGIN
        (* predicate with Boolean var.*)
        IF PP(I, 4) = "T"
          THEN BL := TRUE
        ELSE BL := FALSE;
        R1 := ORD(PP(I, L)) - ORD("A");
        CASE R1 OF
          0 : IF A2 < BL THEN FLAG := 1;
          1 : IF B2 < BL THEN FLAG := 1;
          2 : IF C2 < BL THEN FLAG := 1;
          3 : IF D2 < BL THEN FLAG := 1;
          4 : IF E2 < BL THEN FLAG := 1;
          5 : IF F2 < BL THEN FLAG := 1;
          6 : IF G2 < BL THEN FLAG := 1
        END;
      END
    END
  END
END
7 : IF H2<>BL THEN FLAG:=1
END
END ;

I := I + 1
END ;

(* if all the above conditions are satisfied, *)
(* Tj is firable *)
IF FLAG=0 THEN TT(.K1,J.) := 1
ELSE TT(.K1,J.) := 0
END ;

(* record the values of each PN variable *)
(* in level k1 *)
IF N1 > 0 THEN
(* for integer PN variables *)
FOR I := 1 TO N1 DO
CASE I-1 OF
  0 : QI(.K1,1.) := A ; 1 : QI(.K1,2.) := B ;
  2 : QI(.K1,3.) := C ; 3 : QI(.K1,4.) := D ;
  4 : QI(.K1,5.) := E ; 5 : QI(.K1,6.) := F ;
  6 : QI(.K1,7.) := G ; 7 : QI(.K1,8.) := H
END ;

IF N2 >0 THEN
(* for real PN variables *)
FOR I := 1 TO N2 DO
CASE I-1 OF
  0 : QR(.K1,1.) := A1 ; 1 : QR(.K1,2.) := B1 ;
  2 : QR(.K1,3.) := C1 ; 3 : QR(.K1,4.) := D1 ;
  4 : QR(.K1,5.) := E1 ; 5 : QR(.K1,6.) := F1 ;
  6 : QR(.K1,7.) := G1 ; 7 : QR(.K1,8.) := H1
END ;

IF N3 >0 THEN
(* for Boolean PN variables *)
FOR I := 1 TO N3 DO
CASE I-1 OF
  0 : QL (.K1,1.) := A; 1 : QL (.K1,2.) := B2 ;
  2 : QL (.K1,3.) := C2 ; 3 : QL (.K1,4.) := D2 ;
  4 : QL (.K1,5.) := E2 ; 5 : QL (.K1,6.) := F2 ;
  6 : QL (.K1,7.) := G2 ; 7 : QL (.K1,8.) := H2
END ;

END ;

***********************************************************************
(* NAME : FIRING *)
(* FUNCTION : To fire Tj and get new marking Mkl+1 *)
(* INPUT PARAMETER : J -- the subscript of transition Tj *)
***********************************************************************
PROCEDURE FIRING (VAR J: INTEGER);
VAR I,K,I',J1 : INTEGER ;
BEGIN
  WRITELN (" Firing ");
  IF K0 <> 0 THEN
    BEGIN
    (* check Mkl is in live-loop or not *)
...
R := Q1(. K3. )-1;
WRITE (" M<", K3:2, ",", R:2, "" => "") ;
WRITE (" M<", K0:2, ",", Q1(. K0. )-1:2, "" : "") ;
FOR I := K0 TO K3-1 DO
WRITE (" M<", I:=?, ",", Q1(. I. )-1:2, "" --> "") ;
Writeln (" M<", K0:2, ",", Q1(. K0. )-1:2, "" : "") ;
WRITE (" " );
WRITE (" --- LIVE-LOOPING "'<TERMINAL NODE>'" );
K0 := 0 ; .LVLP := 1 ; K3 := 0
END ;
Q(. K1. ) := J ;
IF N1 > 0 THEN
BEGIN
(* write the values of PB variables in level k1 *)
(*
WRITE (" ");
FOR I := 1 TO N1 DO
CASE I-1 OF
 0 : WRITE(" A =", A:4); 1 : WRITE(" B =", B:4);
 2 : WRITE(" C =", C:4); 3 : WRITE(" D =", D:4);
 4 : WRITE(" E =", E:4); 5 : WRITE(" F =", F:4);
 6 : WRITE(" G =", G:4); 7 : WRITE(" H =", H:4)
END ;
Writeln
END ;
IF N2 > 0 THEN
BEGIN
WRITE (" ");
FOR I := 1 TO N2 DO
CASE I-1 OF
 0 : WRITE(" A' =", A'); 1 : WRITE(" B1 =", B1);
 2 : WRITE(" C1 =", C1); 3 : WRITE(" D1 =", D1);
 4 : WRITE(" E1 =", E'); 5 : WRITE(" F1 =", F1);
 6 : WRITE(" G1 =", G1); 7 : WRITE(" H1 =", H1)
END ;
Writeln
END ;
IF N3 > 0 THEN
BEGIN
WRITE (" ");
FOR I := 1 TO N3 DO
CASE I-1 OF
 0 : WRITE(" A2 =", A2); 1 : WRITE(" B2 =", B2);
 2 : WRITE(" C2 =", C2); 3 : WRITE(" D2 =", D2);
 4 : WRITE(" E2 =", E2); 5 : WRITE(" F2 =", F2);
 6 : WRITE(" G2 =", G2); 7 : WRITE(" H2 =", H2)
END ;
Writeln
END ;
(*
(* write the symbol showing to go to next marking *)
(*

- 157 -
WRITELN ("M<", K1+1:2, "," , Q1(K1+1):2, "> : ");
(* increase the level *)
Q1(K1+1) := Q1(K1+1) + 1;
(* get new marking Mk+1 *)
FOR K:=1 TO L DO
  FOR I:=1 TO M DO
    MR(K1+1, I, K.) := MR(K1, I, K.) - PARC(I, J, K.)
    + TARC(I, J, K.);
    (* execute the action at transition Tj *)
  IF AT(J, 1.) <> "$" THEN
    IF AT(J, 2.) = "": THEN
      BEGIN
        (** for the action with integer PN var. **) 
        R1 := ORD(AT(J, 1.)) - ORD("A") ;
        IF AT(J, 4.) = AT(J, 1.) THEN BEGIN
          (* for the action with form "X:=X+R" *)
          R := 0 ; I1 := 6 ;
          WHILE (AT(J, 1.) <> "$") DO
            BEGIN
              R := R * 10 + ORD(AT(J, I1.)) - ORD("0");
              I1 := I1 + 1
            END ;
          CASE R1 OF
            0 : A := A + R ; 1 : B := B + R ;
            2 : C := C + R ; 3 : D := D + R ;
            4 : E := E + R ; 5 : F := F + R ;
            6 : G := G + R ; 7 : H := H + R
          END ;
        END ELSE BEGIN
          (* for the action with form "X:=R" *)
          R := 0 ; I1 := 4 ;
          WHILE (AT(J, I1.) <> "$") AND
            (AT(J, I1.) <> "$") DO
            BEGIN
              R := R * 10 + ORD(AT(J, I1.)) - ORD("0");
              I1 := I1 + 1
            END ;
          CASE R1 OF
            0 : A := R ; 1 : B := R ;
            2 : C := R ; 3 : D := R ;
            4 : E := R ; 5 : F := R ;
            6 : G := R ; 7 : H := R
          END
        END ;
      END ;
    END ;
END ;
WHILE (AT(J, I1.) = "") DO I1 := I1 + 1;
IF AT(.J,I1.) = 'A' THEN
BEGIN
I1 := I1 + 3;
(* for the second part of 'X:=X+R and Y:=Y+S' *)
WHILE (AT(.J,I1.) <> '') DO I1 := I1 + 1;
R1 := ORD(AT(.J,I1.)) - ORD('A');
IF AT(.J,I1+3.) = AT(.J,I1.) THEN BEGIN
    R := 0; I1 := I1 + 5;
    WHILE (AT(.J,I1.) <> '') DO BEGIN
        R := R * 10 + ORD(AT(.J,I1.)) - ORD('0');
        I1 := I1 + 1
    END;
    CASE R1 OF
        0: A := A + R; 1: B := B + R;
        2: C := C + R; 3: D := D + R;
        4: E := E + R; 5: F := F + R;
        6: G := G + R; 7: H := H + R
    END
END
ELSE BEGIN
    R := 0; I1 := I1 + 3;
    WHILE (AT(.J,I1.) <> '$') DO BEGIN
        R := R * 10 + ORD(AT(.J,I1.)) - ORD('0');
        I1 := I1 + 1
    END;
    CASE R1 OF
        0: A := A + R; 1: B := B + R;
        2: C := C + R; 3: D := D + R;
        4: E := E + R; 5: F := F + R;
        6: G := G + R; 7: H := H + R
    END
END
END
ELSE
IF AT(.J,2.) = '1' THEN BEGIN
(** for the action with real PN var. **)  
I1 := 8;
R0 := 0;
WHILE AT(.J,I1.) <> '.' DO BEGIN
    R := ORD(AT(.J,I1.)) - ORD('0');
    R0 := R0 * 10 + R;
    I1 := I1 + 1
END;
J1 := I1;
I1 := J1 + 1;
WHILE AT(.J,I1.) <> '$' DO BEGIN
    R := ORD(AT(.J,I1.)) - ORD('0');
END
R0 := R0 + EXP((I1-J1) * LN(0.1)) ;
I1 := I1 + 1
END ;
R* := ORD(AT(.J 1.)) - ORD("A") ;
CASE R1 OF
  0 : A1 := A1 + R ;
  1 : B1 := B1 + R ;
  2 : C1 := C1 + R ;
  3 : D1 := D1 + R ;
  4 : E1 := E1 + R ;
  5 : F1 := F1 + R ;
  6 : G1 := G1 + R ;
  7 : H1 := H1 + R
END
ELSE BEGIN
  IF AT(.J,9.)="T" THEN BL := TRUE
  ELSE BL := FALSE ;
  R1 := ORD(AT(.J 1.)) - ORD("A") ;
CASE R1 OF
  0 : A2 := A2 OR BL ;
  1 : B2 := B2 OR BL ;
  2 : C2 := C2 OR BL ;
  3 : D2 := D2 OR BL ;
  4 : E2 := E2 OR BL ;
  5 : F2 := F2 OR BL ;
  6 : G2 := G2 OR BL ;
  7 : H2 := H2 OR BL
END
END ;
FOR I:=1 TO M DO BEGIN
  FOUND := 0 ;
  K := 1 ;
  WHILE (K<=L) AND (FOUND=0) DO
    IF (PARC(.I,J,K.)<>0) OR (TARC(.I,J,K.)<>0)
      THEN FOUND := 1
    ELSE K := K + 1 ;
    IF AP(.I,1.) <> "S" THEN
      IF FOUND = 1 THEN
        IF AP(.I,2.) = "*" THEN
          THEN BEGIN
            (** for action with Boolean PN var. **) 
            R := ORD(AP(.I 6.)) - ORD("0") ;
            R1 := ORD(AP(.I 1.)) - ORD("A") ;
            CASE R1 OF
              0 : A := A + R ;
              1 : B := B + R ;
              2 : C := C + R ;
              3 : D := D + R ;
              4 : E := E + R ;
              5 : F := F + R ;
              6 : G := G + R ;
              7 : H := H + R
            END
          END
        END
      END
    END
  END
END ;
END
ELSE IF AP(I, 2) = '1'
THEN BEGIN
  R := ORD(AP(I, 8)) - ORD('0');
  R' := ORD(AP(I, 1)) - ORD('A');
  CASE R1 OF
    0 : A1 := A1 + R ;
    1 : B1 := B1 + R ;
    2 : C1 := C1 + R ;
    3 : D1 := D1 + R ;
    4 : E1 := E1 + R ;
    5 : F1 := F1 + R ;
    6 : G1 := G1 + R ;
    7 : H1 := H1 + R ;
  END
END
ELSE BEGIN
  IF AP(I, 9) = 'T' THEN BL := TRUE
    ELSE BL := FALSE;
  RL := ORD(AP(I, 1)) - ORD('A');
  CASE RL OF
    0 : A2 := A2 OR BL ;
    1 : B2 := B2 OR BL ;
    2 : C2 := C2 OR BL ;
    3 : D2 := D2 OR BL ;
    4 : E2 := E2 OR BL ;
    5 : F2 := F2 OR BL ;
    6 : G2 := G2 OR BL ;
    7 : H2 := H2 OR BL ;
  END
END
END

**************************************************************************
(* NAME : LEAF *)
(* FUNCTION : To determine the new marking Mk1+l is a *)
(*          leaf (terminal node) or not, *)
(*          if so, determine its type: home-state or *)
(*          live-looping, or # level > E0. *)
(* INPUT PARAMETER : *)
(* OUTPUT PARAMETER : *)
(* J -- subscript of the fired transition Tj *)
(* UU -- 1/0, Mk1+l is an internal node/leaf *)
(* PROGRAM IDENTIFIER : FOUND -- exist k2 and k, so that *)
(*          Mk1+l(k) < Mk2(k) *)
(* FOUND1 -- exist k2 and k, so that *)
(*          Mk1+l(k) > Mk2(k) *)
(* KFOUND -- exist k2, so that *)
(*          Mk1+l = Mk2 *)
(* KFOUND1 -- exist k2, so that *)
(*          Mk1+l > Mk2 (w-symble) *)
**************************************************************************

PROCEDURE LEAF (VAR J, UU : INTEGER);
VAR I,K,FOUND, KFOUND : INTEGER;
QK1 : ARRAY(.1..40,1..9.) OF INTEGER;
BEGIN
  KFOUND := 0 ;  KFOUND1 := 0 ;
  IF WFLAG=1 THEN K2 := 1
  ELSE K2 := K4 := 1 ;
  /*
  /* find whether exist k2 <= k1, that M_{k1+l} = M_k2
  /* and whether exist K4 <= k1, that M_{k1+l} > M_k4
  /*
  */
  WHILE (K2<=K1) AND (KFOUND=0) AND (KFOUND1=0) DO
  BEGIN
    FOUND := 0 ;  FOUND' := 0 ;
    FOR K:=1 TO L DO
      FOR I:=1 TO M DO
        QK1(.I,K.) := 0 ;
    I := 1 ;
    WHILE (I<=M) AND (FOUND=0) DO
      BEGIN
        K := 1 ;
        WHILE (K<=L) AND (FOUND=0) DO
          IF QK(.I,K.) = 0
            THEN IF MR(.K1+1,I,K.) < MR(.K2,I,K.)
              THEN FOUND := 1
              ELSE IF MR'.K1+1,I,K.) > MR'.K2,I,K.)
                THEN BEGIN
                  FOUND1 := 1 ;
                  QK1(.I,K.) := 1 ;
                  K := K + 1
                END
            ELSE K := K + 1
        END;
        IF FOUND = 0
          THEN IF FOUND1 = 0
            THEN BEGIN
              KFOUND := 1 ;  (* found k2 *)
              K0 := K2
              END
            ELSE BEGIN
              KFOUND1 := 1 ;  (* found k4 *)
              K4 := K1
              END
        ELSE K2 := K2 + 1
      END;
    IF KFOUND1 = 0
      THEN FOR K:=1 TO L DO
        FOR I:=1 TO M DO
          QK1(.I,K.) := 0 ;
    FOR K:=1 TO L DO
      FOR I:=1 TO M DO
        QK(.I,K.) := QK(.I K.) + QK1(.I,K.) ;
  END
END ;
(* write the new marking Mkl+1 as a node of the tree *)

FOR K:=l TO L DO
  BEGIN
    WRITE(' ');
    FOR I:=1 TO M DO
      IF WFLAG = 1
        THEN WRITE (MR(.Kl+1,I,K.):2,"" )
      ELSE IF QK(.I,K.) = 0
        THEN WRITE('/MR(.Kl+1,I,K.):"","" )
        ELSE WRITE('"W",""'); (() w-symbol *)
    WRITELN(">")
  END;
  IF KFOUND = 0
  THEN BEGIN
    (* no such k2 or k4 exist, check current *)
    (* # of level exceeds E0 or not *)
    K1 := K1 + 1;
    IF K1 <= E0
      THEN UU := 1 (* not exceeds *)
    ELSE BEGIN
      WRITE (' M<",K1:"",,Ql(.K1.):2,"">');
      WRITE ('--- LEVELS >= ALLOWED ');
      WRITELN (" < TERMINAL NODE >");
      WRITELN ;
      E3 := 1 ;
      K1 := K1 - 1 ;
      UU := 0 (* Mkl+1 is a leaf *)
    END
  END
  ELSE IF K0 = 1
  THEN BEGIN
    (* Mkl+1 is home-state *)
    WRITE (' M<",Kl+1:?,",,Ql(.Kl+1.):1:?,">');
    WRITELN ('--- HOME-STATE < DUPLICATE NODE >');
    K0 := 0 ; K3 := 0 ; HMST := 1 ; UU := 0
  END
  ELSE IF J = L1 + 1
  THEN BEGIN
    K3 := K1 + 1 ; UU := 0
  END
  ELSE BEGIN
    (* Mkl+1 is in a live-loop *)
    WRITE (' M<",Kl+I:"",,Ql(.Kl+1.):2,"">');
    FOR I:=K0 TO K3-1 DO
      WRITE (' M<",K0:2,"",,Ql(.K0.):1:2,""> -->');
      WRITELN (' M<",K0:2,"",,Ql(.K0.):1,"">');
      WRITE ('--- LIVE-LOOPING < DUPLICATE NODE >');
    K0 := 0 ; LVLP:=1 ; UU := 0
  END
END ;

(*********************************************************************** )

- 163 -
(* NAME : LOCK *)
(* FUNCTION : To detect deadlocks and livelocks *)
(* INPUT PARAMETER : *)
(* J -- the subscript of Tj which just fired *)
(* OUTPUT PARAMETER : *)
(* VV -- 0/1, when go back from a leaf may *)
(* find another branch / not *)

PROCEDURE LOCK (VAR J, VV : INTEGER);
VAR I, K : INTEGER;
BEGIN
  Writeln (" LOCK ");
  IF L1 = N THEN BEGIN
    (* : Mk1+1 is a deadlock *)
    Writeln (" M<" , K1 : 2, "," , Q1 (K1 : 1) - 1 : 2 > = M<" );
    Writeln (" --- DEADLOCK < TERMINAL NODE > " );
    DDLK := 1
  END ELSE IF K0 <> 0 THEN IF L1 = N - 1 THEN BEGIN
    L2 := L2 + 1 ;
    IF K1 = K0 THEN IF L2 <> K3 - K0 THEN L2 := 0 ELSE BEGIN
      Writeln (" Mk1+1 is a livelock ");
      Writeln (" M<" , K3 : 2, "," , Q1 (K3 : 1) - 1 : 2 > = M<" );
      Writeln (" K0:2," , Q1 (K0 : 1) - 1 : 2 , " " );
      FOR I := K0 TO K3 - 1 DO BEGIN
        Writeln (" M<" , I : 2, "," , Q1 (I : 1) - 1 : 2 > = " );
        Writeln (" M<" , K0 : 2, "," , Q1 (K0 : 1) - 1 : 2 , " " );
        Writeln (" --- LIVE-LOCK < TERMINAL NODE > " );
        Writeln ; LVLK := 1.
      END;
    END;
  END;
  IF K1 > 1 THEN VV := 0.
  ELSE VV := 1
END;

(* NAME : VARIABLES *)
(* FUNCTION : To assign the old values in lower level *)
(* to the PN variables when tracing back *)
(* from a leaf to find another branch *)

PROCEDURE VARIABLES ;
VAR I : INTEGER ;
BEGIN
  (* for integer PN variables *)
  IF N1 > 0 THEN
    FOR I := 1 TO N1 DO
      ...
CASE I-1 OF
  0 : A := QI(.,K1,1.);  1 : B := QI(.,K1,2.);
  2 : C := QI(.,K1,3.);  3 : D := QI(.,K1,4.);
  4 : E := QI(.,K1,5.);  5 : F := QI(.,K1,6.);
  6 : G := QI(.,K1,7.);  7 : H := QI(.,K1,8.);
END;
(* for real PN variables *)

IF N3 > 0 THEN
  FOR I:=1 TO N2 DO
    CASE J-1 OF
      0 : A1 := QR(.,K1,1.);  1 : B1 := QR(.,K1,2.);
      2 : C1 := QR(.,K1,3.);  3 : D1 := QR(.,K1,4.);
      4 : E1 := QR(.,K1,5.);  5 : F1 := QR(.,K1,6.);
      6 : G1 := QR(.,K1,7.);  7 : H1 := QR(.,K1,8.);
    END;
    (* for Boolean PN variables *)
  END
END;

(* the main program of procedure REACHABILITY *)

BEGIN
  PREP;
  K1 := 1;
  100 : IF GC = 1 THEN RECEIVE; (* to receive new values of var.*)
        CHECK; (* to check token properties *)
        FIND; (* to find firable transitions *)
        L1 := 0; L2 := 0; K0 := 0; K3 := 0; J := 1;
  200 : WRITELN("200: K1=", K1);
        IF TT (.,K1,J.) <> 0 THEN BEGIN
          (* when Tj is firable *)
          (* to firing Tj *)
          FIRING (J); (* to determine type of leaf *)
          IF XX = 1 THEN GOTO 100
            ELSE IF J<N THEN BEGIN
            J := J + 1;
            GOTO 200
            END
          ELSE BEGIN
            (* detect locks *)
            LOCK (J,YY);
            IF YY = 0 THEN BEGIN

-165-
(* trace back *)
K1 := K1 - 1;
FLAG := 0;

WHILE (K1>=1) AND (FLAG=0) DO
BEGIN
    J := Q(.K1.) + 1;
    IF J <= N THEN FLAG := 1
    ELSE K1 := K1-1
END;

IF FLAG = 1
THEN BEGIN
    Ll := 0;
    (* get old values *)
    VARIABLES;
    IF K1 < K4 + 1 THEN
        BEGIN
            FOR K:=1 TO L DO
                FOR I:=1 TO M DO
                    QK(.I,K.) := 0;
            K4 := 0
        END;
    GOTO 200
END
ELSE GOTO 300
END

ELSE BEGIN
    (* when Tj^i is not firable *)
    (* sum unfirable num. of trans. *)
    IF J < N
        THEN BEGIN
            J := J + 1; (* try next transition *)
        GOTO 200
        END
    ELSE BEGIN
        (* when no more transition left *)
        LOCK (J YY); (* detect locks *)
        IF YY = 0 THEN BEGIN
            (* trace back *)
            K1 := K1 - 1;
            FLAG := 0;
            WHILE (K1>=1) AND (FLAG=0) DO
                BEGIN
                    J := Q(.K1.) + 1;
                    IF J <= N THEN FLAG := 1
                    ELSE K1 := K1-1
                    END;
                IF FLAG=1 THEN BEGIN
                    Ll := 0;
                    VARIABLES;
                    IF K1 < K4 + 1 THEN

BEGIN
FOR K:=1 TO L DO
     FOR I:=1 TO M DO
     BEGIN
         QK(I,K) := 0;
         K4 := 0
     END;
END;
GOTO 200
END

BEGIN
300: IF K0 <> 0 THEN
     BEGIN
         R := QL(K3) - 1;
         WRITE("M<"<,K3:2,"<,R:2,"=",">
             "");
         WRITE("M<"<,K0:2,"<,QL(K0)-1:2,">
             ");
         FOR I:=K0 TO K3-1 DO
             WRITE("M<"<,I:2,"<,QL(I)-1:2,"->")
         WRITELN("M<"<,K0:2,"<,QL(K0)-1:2,">");
         WRITE("<--- LIVE-LOOPING < TERMINAL NODE >")
         K0 := 0;
         K3 := 0;
         LVLP := 1
     END;
     (*
     printout the properties of the PN
     *)
     WRITELN;
     WRITELN("THE REACHABILITY ANALYSIS RESULTS :");
     IF SAF = 1 THEN WRITELN("<1> THE NET IS NOT SAFE.");
     ELSE IF E3 = 0 THEN
         WRITELN("<1> THE NET IS SAFE.");
     ELSE
         WRITELN("<1> THE SAFENESS IS NOT DETECTED.");
     WRITELN;
     IF E3 = 0 THEN
         WRITELN("<2> THE MAX NUMBER OF TOKENS IN EACH PLACE :");
     ELSE
         WRITELN("<2> THE MAX NUMBER OF TOKENS IN EACH PLACE ">=");
         FOR K:=1 TO L DO
             WRITELN("FOR COLOUR<,K:1,<,BD(.K):2>");
             IF CFLAG = 0
             THEN IF E3 = 0 THEN
                 BEGIN
                     WRITELN("<3> THE NET IS CONSERVED BY :");
                     FOR K:=1 TO L DO
                         WRITELN("FOR COLOUR<,K:1,<,CS(.K):2>");
                     WRITELN;
                 END
             ELSE
                 WRITELN("<3> THE CONSERVATION IS NOT DETECTED.");
             END
             END
             ELSE
             WRITELN("<3> THE NET IS CONSERVED BY :");
             FOR K:=1 TO L DO
                 WRITELN("FOR COLOUR<,K:1,<,CS(.K):2>");
             WRITELN;
             END
         END
     END
ELSE WRITELN (" <3> THE NET IS NOT CONSERVED .") ;
WRITELN (" <4> EXECUTION TERMINALS :");
IF DDLP=0 THEN
  WRITELN (" NO DEADLOCKS .");
ELSE
  WRITELN (" THE NET HAS DEADLOCK< S>.");
ENDIF

IF LHLP=0 THEN
  WRITELN (" NO LIVE-LOCKING .");
ELSE
  WRITELN (" THE NET HAS LIVE-LOCK< S>.");
ENDIF

IF HMST=0 THEN
  WRITELN (" M0 IS NOT A HOME-STATE .")
ELSE
  WRITELN (" M0 IS A HOME-STATE .")
ENDIF

WRITELN (" "
WRITELN (" * END OF REACHABILITY ANALYSIS *");
WRITELN (" "

END ;

******************************************************************************

(NAME : INCIDENCE
FUNCTION : To find the place and transition invariants
of the PN
PROGRAM IDENTIFIER :
FFLAG -- 0/1, when procedure OMAGA/FAI calls SOLUTION
PROGRAM ARRAYS :
CM -- 40x40x9, CM = TARC - PARC
CT -- 40x40x9, the transform of CM
X -- 40, solution of CM.X=0 or CT.X=0
PROCEDURES CALLED : OMAGA, FAI

PROCEDURE INCIDENCE ;
VAR CM, CT : CB ;
X : CF ;
I,J,K : INTEGER ;
V,L1,S,T : INTEGER ;
FOUND, FLAG, FFLAG : INTEGER ;

******************************************************************************

(NAME : SOLUTION
FUNCTION : To do the elementary operation for a
MxNxL matrix (B5)
PROGRAM VARIABLES :
M1 -- integer, MIN.(MM,NN)
LL -- integer, rotation variable
TEMP -- integer, used as a temporary variable
PROGRAM IDENTIFIER :
KFLAG -- 1/0, column NO.(k) > total row no.(MM)/not
PROGRAM ARRAYS :
Y(I) -- 1x40, 1/0, I row has zero-element / not
Z(K) -- 1x40, 1/0, K-column, from K-row has no

******************************************************************************
PROCEDURE SOLUTION (VAR B5 : CA ; VAR WW : CF ;
VAR MM, NN, HH, HL : INTEGER) ;

VAR I, IL, J, K, FOUND, FLAG, KFLAG : INTEGER ;
T TEMP, LL : INTEGER ;
Y,Z : ARRAY (.1..40.) OF INTEGER ;
INDEX, M1 : INTEGER ;

BEGIN
  WRITELN( " SOLUTION ");
  (* get M1 = MIN (MM, NN) *)
  IF MM < NN
    THEN M1 := MM
    ELSE M1 := NN ;
  (* initialize Z(.I.) , HL *)
  FOR I:=1 TO NN DO
    Z(.I.) := 0 ;
  INDEX := 0 ; HL := 0 ;
  (*)
  (* elementary operation *)
  (*)
  FOR K:=1 TO NN DO
    BEGIN
      (* for column k, initialize row i *)
      FOUND := 0 ; KFLAG := 0 ;
      IF K=1
        THEN I:=K
      ELSE BEGIN
        LL := 1 ; FLAG := 0 ;
        WHILE (LL<K) AND (LL<=MM-HL) AND (FLAG=0) DO BEGIN
          IF Z(.LL.) =1
          THEN IF B5(.LL,K.) <> 0 THEN BEGIN
              BEGIN
                FLAG := 1;
                I := LL
              END
            ELSE LL:= LL+1
          END ;
          IF FLAG = 0 THEN IF K <= MM THEN I:= K
        ELSE KFLAG := 1 ;
      END ;
    END ;
END ;
IF KFLAG = 0 THEN
(* find non-zero element in kth column : B5(index,k)<>0 *)
WHILE (I<=MM-HL) AND (FOUND=0) DO
  IF B5(.I,K,) <> 0
    THEN BEGIN
      FOUND := 1; INDEX := I;(* found INDEX *)
      IF B5(.I,K,) < 0 THEN
        (* change this non-zero element to positive *)
        FOR J:=K TO NN DO
          B5(.I,J,) := -B5(.I,J,)
      END
    ELSE I := I + 1;
    IF FOUND = 1
      (* kth column has non-zero element *)
      THEN BEGIN
        IF K <> MM
          THEN BEGIN
            (* exchange INDEX row and kth row *)
            IF INDEX <> K
              THEN FOR J:=K TO NN DO
              BEGIN
                T := B5(.K,J,)
                B5(.K,J,) := B5(.INDEX,J,)
                B5(.INDEX,J,) := T
              END
            (* purge other elements in kth row *)
            FOR I:=1 TO K-1 DO
              IF Z(.I,) = 1
                THEN BEGIN
                  TEMP := B5(.I,K,)
                  T := B5(.K,K,)
                  FOR J:=K TO NN DO
                    B5(.I,J,) := B5(.I,J,)*T -
                                 TEMP*B5(.K,J,)
                END
            END
          END
        ELSE BEGIN
          (* for MM<NN and K>MM *)
          (* purge other elements in INDEX column *)
          FOR I:=INDEX+1 TO MM DO
            BEGIN
              TEMP := B5(.I,K,)
              T := B5(.K,K,)
              FOR J:=1 TO NN DO
                B5(.I,J,) := B5(.I,J,)*T - TEMP*B5(.K,J,)
            END
          END
        END
      END
      (* move INDEX row to the bottom, each *)
      (* of INDEX+1 to MM rows move up 1 row *)
      FOR J := 1 TO NN DO
BEGIN
T := B5(INDEX,J) ;
FOR I:=INDEX+1 TO MM DO
  B5(I-1,J.) := B5(I,J) ;
  B5(MM,J) := T
END;
   HL := HL + 1
END

ELSE Z(.K.) := 1
(* after each elementary operation, write the matrix *)
(* WRITELN("Z",K:=",":=:Z(.K.)") *)
(* WRITELN("MATRIX D",K:2) *)
(* FOR I:=1 TO MM DO *)
(* BEGIN *)
(* FOR J:=1 TO NN DO *)
(* WRITE(B5(I,J))*)
(* WRITE(B5(I,J):') *)
(* WRITELN *)
(* END *)
(* WRITELN *)
END;
(* count the total number of zero-rows *)
HL := 0;
FOR I:=1 TO MM DO
BEGIN
  J := 1; FLAG := 0;
  WHILE (J <= NN) AND (FLAG = 0) DO
    IF B5(I,J.) <> 0
      THEN FLAG := 1
      ELSE J := J+1;
    IF FLAG = 0
      THEN BEGIN
        Y(.I.) := 1;
        HL := HL + 1
      END
      ELSE Y(.I.) := 0
END;
(* count the total number of zero-columns *)
HH := 0;
FOR K:=1 TO NN DO
  IF Z(.K.) = 1
    THEN BEGIN
      HH := HH + 1;
      WW(HH) := K
    END;
  IF PFLAG=1
    THEN FOR I:=HH+1 TO HH+(NN-MM) DO
      WW(I.) := MM+I-HH ;
(* move all the zero-rows to the bottom *)
FOR K:=1 TO MM-1 DO
  FOR I:=1 TO MM-1 DO
    IF Y(.I) = 0
      THEN IF Y(I+1) = 0
        THEN BEGIN
          *
Y(.I.) := 0 ; Y(.I+1.) := 1 ;
FOR J:= 1 TO NN DO
BEGIN
  T := B5(.I+1.,J.) ;
  B5(.I+1.,J ) := B5(.I J ) ;
  B5(.I.,J ) := T
END
END ;
(* move up the non-zero rows who has less no. zero-element *)
FOR K:=1 TO M1 DO
  IF B5(.K,K.,) = 0 THEN BEGIN
    J := K+1 ; FOUND := 0 ;
    WHILE (J<=NN) AND (FOUND=0) DO
      IF B5(.K,J.,) = 0 THEN BEGIN
        I := K + 1 ; FLAG := 0 ;
        WHILE (I<=MM) AND (FLAG=0) DO
          IF B5(.I,J.,) <> 0 THEN BEGIN
            FLAG := 1 ;
            FOR LL:=1 TO NN DO
              BEGIN
                T := B5(.K,LL.) ;
                B5(.K,LL.) := B5(.I,LL.);
                B5(.I.,LL.) := T
              END
          END ELSE I := I + 1
        END ELSE FOUND := 1
      END
    END
  END
END ;
(* write the final equivalent matrix *)
WRITELN(’MATRIX D’)
(*) FOR I:= 1 TO MM DO
(*) BEGIN
(*) (* FOR J:= 1 TO NN DO *)
(*) (* WRITE(B5(.I,J.):4) *)
(*) (* WRITELN *)
(*) END
(*) WRITELN (’ MAX ORDER = ’,MM-HL :2)
END ;
(* NAME : OMGA *)
(* FUNCTION : To find the transition invariants by solving *)
(* CM.X=0 *)
(* PROGRAM VARIABLES : *)
(* V -- integer, total number of solution groups *)
(* H -- integer, total number of 0-columns in AAl *)
(* H1 -- integer, total number of 0-rows in AAl *)
(* PROGRAM IDENTIFIER : *)
(* XFLAG -- 0/1, solution X(INDEX) = integer / not *)
(* PROGRAM ARRAYS : *)
BEGIN
  FOR K2:=1 TO L DO
    BEGIN
      (* for each k, get AAl from AA *)
      FOR I:=1 TO M DO
        FOR J:=1 TO N DO
          AAl(I,J) := AA(I,J,K2);
          (* call SOLUTION to do elementary operations on AAl *)
          SOLUTION('AAl,W,M,N,H,H1');
          (* WRITELN('** * COLUMN # H=',H,') 0-ROW # H1=',H1) *)
          IF M-H >= N
            THEN WRITELN
              ('** * THERE IS NOT ANY TRANSITION INVARIANT FOR THIS NET **')
            ELSE BEGIN
              (* get the total number of possible 0/1 solutions *)
              V := 1;
              K := 0;
              REPEAT
                V := V * 2;
                K := K + 1
              UNTIL K = H;
              (* WRITELN('V=',V) *)
              I2 := 0;
              FOR LL:=1 TO V-1 DO
                (* assign 0/1 to enrich arbitrary Xi *)
                (* in the LLth solution *)
                BEGIN
                  J := 1;
                  S := LL;
                  REPEAT
                    X(I,W(J)) := S MOD 2;
                    S := S DIV 2;
                    J := J + 1
                  UNTIL J = H + 1;
                  (* FOR J:=1 TO H DO
                    IF W(J) <= N *)
                  END;
                  (* THEN *)
                  (* WRITELN('X',W(J),':',X(I,W(J))) *)
      END
    END
END

- 173 -
L1 := M-H1;  XFLAG := 0;

(*
  NON-ZERO ROW # M-H1 = ^L1:2
*)

WHILE (L1>0) AND (XFLAG=0) DO

BEGIN
(*
  find first non-0-element
*)

  (*
   in the last non-0-row
  *)

  J := 1;  FOUND := 0;  T := 0;

  WHILE (J<=N) AND (FOUND=0) DO

    IF AAL(L1,J) <> 0

      THEN BEGIN

        FOUND := 1;

        INDEX := J  (* find the column # *)

      END

    ELSE J := J+1;

END

(*
  get the value of X(INDEX)
*)

IF INDEX < N THEN

BEGIN
FOR J:=INDEX+1 TO N DO

  T := T-AAL(L1,J) * X(J);

  X(INDEX,) := T DIV AAL(L1,INDEX,);

  IF X(INDEX) * AAL(L1,INDEX,) <> T

    THEN XFLAG := 1;

END

ELSE X(INDEX,) := 0;

L1 := L1-1

END

(*
  to find only integer solutions
*)

IF XFLAG = 0 THEN

BEGIN
(*
  to determine whether this solution is 0/1 or not
*)

  FLAG := 0;

  I := 1;

  WHILE (I<=N) AND (FLAG=0) DO

    IF (X(I,1)<>0) AND (X(I,1)<1)

      THEN FLAG := 1

    ELSE I := I+1;

(*
  if it is 0/1 solution, printout the invariant
*)

IF FLAG = 0 THEN

BEGIN

I2 := I2 + 1;

FOR I1:= 1 TO N DO

  X(I1:3,:) = :4,X(I1:1)

  writeln;

  writeln;

  WRITE( TRANSITION INVARIANT , I2:2, 

  WRITE( 

  NN := 0;

  FOR I:=1 TO N DO

    IF X(I,1)=1 THEN

      BEGIN

      WRITE("T":I:1);

      NN := NN + 1

      END;

      writeln;

  - 174 -
WRITE('')
WRITE('M')
FOR I:=1 TO NN DO WRITE('---')
WRITELN('>')
END
END
END
END
END
END

(* *******************************************************
(* NAME : FAI
(* FUNCTION : To find the place invariants by solving
(*      CT.X=0
(*
(* INPUT PARAMETER :
(*      BB -- 40X40X9, integer, BB=CT
(*
(* PROGRAM VARIABLES :
(*      H -- integer, total number of 0-columns in BB1
(*      H1 -- integer, total number of 0-rows in BB1
(*
(* PROGRAM ARRAYS :
(*      BB1 -- 40X40, integer, BB1= BB for one k
(*      W -- 1X40, integer, w(i) is original # of ith 0-row
(*      X -- 1X40, integer, the solution
(*
(* PROCEDURE CALLED : SOLUTION
(* *******************************************************

PROCEDURE FAI (VAR BB : CB);
VAR BB1 : CA ;
W,X : CF ;
SUM, INDEX, XFLAG :INTEGER ;
I,J,I1,I2,K2,LL,H,H1 : INTEGER ;
BEGIN
WRITELN('FAI')
WRITELN('********************');
WRITELN('PLACE INварIANTS')
WRITELN('********************');
WRITELN;
FOR K2:=1 TO L DO
BEGIN
(* get BB1, BB1 = BB(i,j,k2) *)
FOR I:=1 TO N DO
FOR J:=1 TO M DO
BB1(I,J,) := BB(I,J,K2) ;
(* call SOLUTION to do elementary operations *)
SOLUTION 'BB1,W,N,M,H,H1')
IF N-H >= M THEN WRITELN
("* THERE IS NOT ANY PLACE INварIANT FOR THIS NET*")
ELSE BEGIN
(* to count the total number of solutions *)
V := 1;
K := 0;
REPEAT
---
V := V * 2;
K := K + 1;
UNTIL K = H;
I2 := 0;
FOR LL:=1 TO V-1 DO
BEGIN
(* assign 0/1 to arbitrary Xi in LL th solution *)
J := 0;
S := LL;
REPEAT
J := J + 1;
X(.W(.J.)) := S MOD 2;
S := S DIV 2
UNTIL J=H;
FOR J:=1 TO H DO
BEGIN
(* bottom-up, to find LLth solution *)
LL:= N-H'; XFLAG := 0;
WHILE (LL>0) AND (XFLAG=0) DO
BEGIN
(* to find the first non-0-element in the last non-0-row *)
T := 0;
J := 1; FOUND := 0;
WHILE (J<=M) AND (FOUND=0) DO
IF BB1(.LL,J.) <> 0
THEN BEGIN
FOUND := 1;
INDEX := J (* found column # INDEX *)
END
ELSE J := J+1;
(* to get the value of X(INDEX) *)
IF INDEX < M THEN BEGIN
FOR J:=INDEX +1 TO M DO
T := T-BB1(.LL,J.) * X(.J);
X(.INDEX.) := T DIV BB1(.LL,INDEX.);
IF X(.INDEX.) * BB1(.I',INDEX.) <> T
THEN XFLAG := 1
END
ELSE X(.INDEX.) := 0;
END
(* consider only integer solution *)
IF XFLAG = 0 THEN BEGIN
(* check LLth solution is only 0/1 or not *)
FLAG := 0; I := 1;
WHILE (I<=M) AND (FLAG=0) DO
IF (X(.I.) <> 0) AND (X(.I.) <> 1)
THEN FLAG := 1
ELSE I := I + 1;
(* if only 0/1, printout the place invariant *)
IF FLAG = 0
THEN BEGIN

- 176 -
I2 := I2 + 1;
FOR I1 := 1 TO M DO
  WRITELN(" X("', I1:3, ',") =", 4, X (', I1:1, ') *
  WRITELN;
  WRITELN(" PLACE INVARIANT ', I2:2, ', "");
  WRITE (';
  FOR I := 1 TO M DO
    BEGIN
      FLAG := 0;
      J := I + 1;
      WHILE (J <= M) AND (FLAG = 0) DO
        IF X (', J:1, ') <= 0
          THEN FLAG := 1
          ELSE J := J + 1;
        IF X (', I:1, ') <= 0
          THEN IF FLAG = 1
            THEN WRITE (' M (P':4, ', I:1, ', ') +':3)
            ELSE WRITE (' M (P':4, ', I:1, ', ') :2)
          END;
    END;
    SUM := 0;
    FOR I := 1 TO M DO
      SUM := SUM + X (', I:1, ') * P (', I, K2:1, ') ;
    WRITELN(" =':3, SUM:2)
  END;
END;
END

END;

(*
(* the main program of procedure INCIDENCE *)
(*
BEGIN
  FFLAG := 0;
  WRITELN(" INCIDENCE");
  (* * get CM, CM = TARC - PARC *)
  FOR I := 1 TO M DO
    FOR J := 1 TO N DO
      FOR K := 1 TO L DO

  (* WRITELN(" " CM")
  (* FOR I := 1 TO M DO
    (* BEGIN
      (* WRITE (';
        (* FOR J := 1 TO N DO
          (* FOR K := 1 TO L DO
            (* WRITE (CM (', I, ', J, ', K:1, ') :4)

      (* WRITELN
      (* END
    (* to get CT, CT = transform of CM *)
    FOR K := 1 TO L DO
      FOR I := 1 TO N DO
        FOR J := 1 TO M DO
          CT (', I, ', J, ', K:1, ') := CM (', I, ', J, ', K:1, ') ;

- 177 -
(* WRITELN('" CT" ) *)
(* FOR I:=1 TO N DO *)
(* BEGIN *)
(* WRITE('"" ) *)
(* FOR J:=1 TO M DO *)
(* FOR K:=1 TO L DO *)
(* WRITE (CT(I,J,K) :" ) *)
(* WRITELN *)
(* AND *)
(* OMAGA (CM) ; (* call OMAGA to find transition invariants *)
FFLAG := 1 ;
FAI (CT) ; (* call FAI to find place invariants *)
WRITELN(" --------------- END OF INCIDENCE ANALYSIS ---------------");
WRITELN(" -------------------------------");
END;
(* ---------------------------------------------------------------)
(* NAME : ANEW *)
(* FUNCTION : contain the options of process ANALYSIS *)
(* ---------------------------------------------------------------)
PROCEDURE ANEW ;
BEGIN
WRITELN(" < MODULE ANALYSIS>");
WRITELN(" -----------------------------------------------");
WRITELN(" * DO YOU WANT TO DO : ");
WRITELN(" * 1. REACHABILITY ANALYSIS ");
WRITELN(" * 2. INCIDENCE ANALYSIS ");
WRITELN(" -----------------------------------------------");
WRITELN(" * ENTER YOUR SELECTION * ");
WRITE (" * HIT ANY DIGIT OTHER THEN 1 AND 2 TO END ");
WRITELN(" ANALYSIS ");
END;
(* ---------------------------------------------------------------)
(* NAME : AINTO *)
(* FUNCTION : To display the options on the terminal and *)
(* get the user's selection *)
(* ---------------------------------------------------------------)
PROCEDURE AINTO ;
BEGIN
T := 'N';
WHILE T <> 'Y' DO BEGIN
ANEW; (* call ANEW to display options *)
READLN;
READ (R); (* accept user's selection *)
WRITELN("<',R:1,'> OK ? --- Y OR N'");
READLN;
READ (T) (* accept user's Y/N decision *)
END
END;
(* the main program of procedure ANALYSIS *)
(* *)
BEGIN
    AINTO ;  
    (* call AINTO to display options and get user's selection *)
    (* )
    .WHILE (R>0) AND (R<3) DO
        BEGIN
            IF R=1 THEN REACHABILITY
                ELSE IF R=2 THEN INCIDENCE ;
            AINTO (* call AINTO to get next selection from user *)
            END ;
            WRITELN("************************************************************************") ;
            WRITELN(" * END OF ANALYSIS **") ;
            WRITELN("************************************************************************")
            END ;
        END ;